

AN EXPERT SYSTEM FOR SHORT-TERM WEATHER FORECASTING

23

by

David Charles Willett

B.S., Georgia Institute of Technology, 1977

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

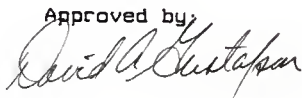
MASTER OF SCIENCE

Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by



LL
21068
. H4
Cmcc
1788
10055
C 2

TABLE OF CONTENTS

A11208 232238

Chapter	Page
1 - Introduction	1
2 - Overview of Artificial Intelligence	9
3 - Overview of Expert Systems	26
4 - The Investigation - Initial Activity	35
5 - System Development and Testing	53
6 - Extensibility Analysis	88
7 - Conclusions	100
8 - Recommendations for Further Work	104
Bibliography	106
Appendix I	
Appendix II	

LIST OF FIGURES

Figure	Page
1 - Example of a Semantic Network	17
2 - Internal Representation of a Rule	48
3 - Internal Representaion of Prompts and Translations	49
4 - Cloud Structure of a Typical Norwegian Wave Cyclone	59
5 - Meteo.kb Decision Tree (12 Plates)	63
6 - Structural Diagram of the Flawed Rule Base	78

Acknowledgements

I would like to acknowledge the support of my management at AT&T Technologies as well as the support of the AT&T Summer-On-Campus program which made this project possible. I would also like to acknowledge the support and encouragement provided by my major professor, Dr. David A. Gustafson.

Further, I would like to acknowledge the support of my wife, Jackie, who took care of all the details on the home front so that I could concentrate on working toward my degree. Finally, I would like to thank my children, Kristen and Charles Michael for giving up their playtime with daddy so that he could finish his report.

CHAPTER 1

PROJECT OVERVIEW

This project studies the technology of Expert Systems. In particular, it focuses on Expert Systems which are hosted on microcomputers. Dramatic advances in hardware technology have made it possible for comparatively inexpensive microcomputers to support software that was once restricted to costly minicomputers and mainframes. The increased power of these new microcomputers makes Expert System technology available to the commercial engineering community. This project examines the newly available technology and considers whether it is adequate for the types of heuristic problems encountered in a commercial engineering environment. It makes this investigation by using a commercially available shell to build a microcomputer based Expert System to forecast short-term local weather. Short-term weather forecasting is typical of the heuristic problems encountered in a commercial engineering setting. In addition, the shell is analyzed to estimate its extendability.

Expert Systems are computer programs which emulate the behavior of human experts by using an inference mechanism to operate on a stored knowledge base. These

systems show great promise for making the expertise of scarce human specialists more generally available. While expert systems are not yet ready to replace human experts, they can help non-expert specialists to perform at expert level.

Despite their promise, Expert Systems have not spread into the general engineering or technical community but have remained concentrated in specialized research circles. A significant limitation on their spread has been that systems sophisticated enough to handle problems of general interest have required prohibitively expensive high performance hardware. Another impediment has been that Expert Systems are usually implemented in the esoteric "AI languages" such as LISP and Prolog. The steady advance of computer technology has brought powerful machines to the desktop. The availability of these low cost machines has spawned interest in Expert System development tools (sometimes called "shells") which advertise that they have brought Expert System development within reach of the ordinary programmer.

There remain several open questions regarding the actual capabilities of these shells. While it is clear that the shells can illustrate the promise of Expert

System technology by supporting limited "sample" systems, it is less certain that they possess the sophistication and extensibility to handle problems of general interest.

The specific questions I will investigate are:

1. Can such a tool support the development of Expert Systems that are powerful enough to handle problems of "real world" sophistication?
2. What design issues become significant for the development of these systems?
3. Under what circumstances is the development of a small Expert System justified from an economic point of view?
4. If extensions to the tool become necessary, can they be made in a cost-effective manner?

To answer these questions, it is necessary to develop an Expert System that tackles a problem typical of the kind that an Industrial Expert System would be built to solve. Such problems arising in an industrial environment possess unique characteristics. The

problems would not be trivial or deterministic. Trivial or deterministic problems are more efficiently attacked with algorithmic decision trees which are cheaper and easier to develop than Expert Systems. On the other hand, the problem would not be intractable. A recognized body of expertise or set of heuristics which yield acceptable solutions would exist. The heuristics would not necessarily be sufficient for all instances of the problem (for example, there are cases in which physicians fail to diagnose an illness) but the heuristic set would be sufficiently robust to justify the effort required to capture it. Further, it would be the case that the reasoning or inference system used is sufficiently arcane that it is more cost effective for the user to consult the Expert System than it is for him (or her) to master the inference method directly through training. The last characteristic eliminates problems that reduce to a set of "normal" cases which can be handled by algorithmic decision trees and a small, finite set of exception cases that arise rather infrequently.

Expert Systems developed for the industrial or commercial environment will have specific design goals dictated by the nature of their target user populations.

While the particular design goals of these systems will, of course, be system specific, there will be a set of goals that is common to all such products. The design goals for my system are my estimation of the goals in this common set.

The system must be supported on a popular architecture (such as an MS-DOS machine having a minimum of 512 kilobytes [K] of RAM). This goal is driven by the fact that such machines are somewhat ubiquitous. While the maximum memory available to MS-DOS is 640K, the 512K configuration is more common. The primary purpose of an expert system is to make expertise available to a large user population. It stands to reason that such systems would be of limited usefulness if they required specialized or inaccessible hardware to run.

The system should support multiple (at least two) levels of expertise. This goal is again a reflection of the original purpose of expert systems. The idea is to make scarce expertise available to less specialized individuals. If the system could not function adequately except with users having a level of expertise equal to its own, then what purpose is served by encapsulating that expertise within a computer system?

Clearly, the system must be able to interact with users on a level below that which the system represents. Practically, this goal translates into being able to glean meaningful data from input expressed in simplified terminology. Conversely, the system must be able to state its conclusions in a way that is useful to the non-expert specialist.

The system must be able to grow in sophistication. Heuristic methods are rarely static. As new measurement technology becomes available, better quality and/or different inputs are available for analysis. In the specific case of meteorology, localized phenomena (such as mountains, lakes, urban heat islands and the like) may have significant effect on the prevailing local weather. It is possible to develop heuristics to account for these effects. The system's knowledge base should be extensible so that these new heuristics can easily be added.

The system must possess an comfortable user interface. This goal is almost archetypical. This goal states that the user interface should lend itself to customization to fit the needs of specific user communities.

The system should conduct consultations in an English-like vocabulary. For design purposes, the sophistication level of the vocabulary was specified as that of the average American adult. Of course, the more general form of this goal is that the system have a natural language vocabulary but the target user population for my system was restricted to native speakers of English.

Finally, the system should be portable and generally extensible. It is reasonable to expect that specific user populations (such as the engineering community) will have substantial installed bases of custom hardware which are tailored to functions they already perform. The system should be able to run on that existing hardware with a minimum of modification. Further, since Expert Systems are domain specific, it would be unusual if a general purpose tool were optimum for a particular user population. The tool then, should comfortably support tailoring.

This report documents the investigation. It begins with some background on the field of Artificial Intelligence from which Expert Systems developed. This discussion is followed by a more detailed treatment of the technologies specific to Expert Systems. The

investigation itself is then described, beginning with the selection of short-term weather forecasting as a test problem and ending with a discussion of the mechanics of building an "industrial strength" Expert System. The investigation also included an extensibility analysis of the particular shell selected which is described following the discussion of system development. The report concludes with an assessment of the state of this development technology.

CHAPTER 2

OVERVIEW OF ARTIFICIAL INTELLIGENCE

Expert Systems are a product of the subdiscipline of Computer Science known as Artificial Intelligence (AI). Since a basic understanding of the fundamental concepts involved in AI is necessary to appreciate Expert Systems, it is appropriate to begin with some introductory material concerning the nature of AI. Please note that this material will cover the field superficially and will focus on those concepts which are most applicable to Expert Systems. The reader is referred to the bibliography included at the end of this paper for more detailed material on AI.

Artificial Intelligence is a specialty of Computer Science that is concerned with the development of "intelligent" machines. A precise definition of the term "intelligent" (or of the noun "intelligence") is something which has eluded researchers in the fields of Psychology and Computer Science for many years. Much of the early work in the field of AI focused on characterizing the properties of an "intelligent" versus those of a "non-intelligent" machine. The well respected British mathematician Alan Turing is credited with devising the first recognized criterion for

characterizing an "intelligent" system.

Turing's test [7], which today bears his name, is a blind test in which a human volunteer is permitted to interact with the system being tested and a human control. Since it is a blind test, the volunteer is not permitted any communication with either the system being tested or the human control except that provided by the communication terminal. Thus, the volunteer has no way of knowing whether he or she is conversing with the human control or with the system being tested. The system being tested is judged "intelligent" if the volunteer is unable to distinguish between its responses and those of the human control. In other words, an "intelligent" system is one that exhibits behavior patterns similar to those that humans exhibit in the same situation. To put it another way, "Intelligence is as intelligence does."

Subsequent work (namely the development of the program ELIZA [7]) showed that the Turing test is less than satisfactory as a universal standard for intelligence. ELIZA showed that it was possible to fool the human volunteer into believing that the machine's responses were the result of intelligence when in fact they were not. ELIZA was a program developed at the

Massachusetts Institute of Technology (MIT) that purported to simulate the behavior of a non-directive therapist. Numerous trials convinced researchers that subjects who did not know they were conversing with a machine (the "blind test" condition of the Turing test) could not distinguish between ELIZA and a human therapist. Unfortunately, ELIZA was not pursuing a goal. Instead, the program was instructed to pattern match on key words (character strings) in the patient's responses and frame its response based on those key words. The entire action of the program was to generate responses. As an example of this process, consider the following:

Human: I never really knew my mother.

Program: Tell me more about your childhood.

The program simply pulled that response from a stored lookup table associated with the keyword MOTHER. When it was unable to find any keywords, ELIZA resorted to a non-committal response such as "Please continue."

It was clear from ELIZA and programs like it that clever programming rendered the Turing test or any test based on human perception of program behavior unreliable

as a test for intelligence. Further, it was becoming clear that no universally accepted definition of "intelligence" either artificial or natural, existed. The best that was available was the informal notion that computer systems (or more accurately programs) were "intelligent" if they produced results or interacted with their environment in ways similar to those a human would. The results produced had to be essentially the same as those a human would produce if asked to solve the same problem. This notion of "intelligence" has continued virtually unchanged to the present. It is in the context of this "definition" that much of the current work in AI including Expert Systems, should be viewed. More formal definitions of intelligence both natural and artificial, await a clearer understanding of brain physiology in humans.

Though it is often on the leading edge of Computer Science, AI is not a new discipline. Its beginnings can be traced to a conference held at Dartmouth University in 1956 [7]. The meeting has since come to be known simply as "The Dartmouth Conference". The participants met to discuss whether computers could be made to "think". At the time the principal "higher" activity under consideration was machine translation of natural

languages. It was widely believed that machine translation was a trivial problem requiring only a knowledge of the constituent grammars and lexicons and some simple programming. It was assumed that natural languages were similar to machine languages and that the job of translation was straightforward. In fact, many were predicting that automated natural language translators were just a few short years away.

It quickly became apparent that the task of understanding natural language was significantly more complex than had first been presumed. The major obstacle turned out to be the lack of sufficiently powerful tools for representing the semantics or meaning of natural language statements. Machine languages are constrained so that contextual ambiguity is minimized or eliminated. Natural languages have no such constraints. To illustrate the difficulty, consider the following example:

John gave Mary a book. She put it on the shelf.

Joe said, "Give it to me!"

To the human reader it is quite clear that Joe is

requesting that Mary hand the book to him instead of placing it on the shelf. To a computer, the meaning is not nearly so precise. The pronoun "it" could refer to the book or to the shelf, or to something else altogether. The computer has no way of knowing which meaning to apply. The human reader, on the other hand, can bring his previous experiential and contextual knowledge to bear on the problem and can thus resolve the ambiguity concerning the word "it". The challenge to researchers in AI then, was to develop techniques which allow such contextual and experiential knowledge to be represented internally within a computer system so that it is available to a program.

One such technique developed was the use of "frames" [2]. Frames were first proposed by Marvin Minsky of MIT and are actually thought templates which place a word within an experiential context. To illustrate, consider the word "chair". To a human, the meaning of the word "chair" connotes, not a dictionary definition, but an experiential reference to a certain type of object. Within this experiential reference the terms "a soft chair" and "a large chair" have meaning. This sort of experiential templating can be represented by a structure such as the one shown below:

CHAIR

COMPONENTS: Back [Required]

Seat [Required]

Arms [Optional]

PURPOSE: For humans to sit on the seat

DIMENSIONS: Length (1-3 ft.), Height of Seat

(18 in.),

Height of Back (18-34 in.), Width

(20 in.)

COMPOSITION: Hard (wood, steel, plastic)

Soft (cloth, vinyl)

NO. OF LEGS: 4 [Required]

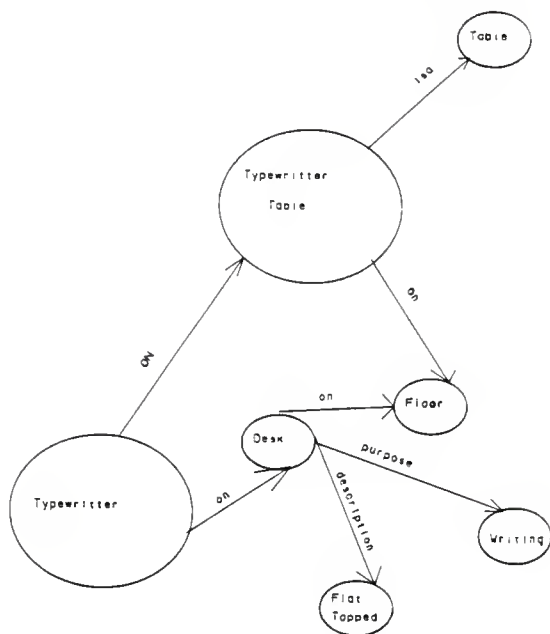
It should be clear that equipped with such a structure, a computer program would be able to assign the expected interpretations to the phrase "a soft chair".

Minsky's work with frames was extended to the concept of "scripts" [2]. Scripts are similar to frames except that they are time variant instead of static. They are intended to represent normal sequences of events or scenarios concerning everyday activities. Scripts then, provide a technique for representing "default" information about sequences of events. Humans would normally supply such "default" information from their own personal experiences in similar situations. One popular example of script representation is the normal sequence of events that occurs when a patron dines at a restaurant.

Another knowledge representation technique is the

Semantic Network [10]. Semantic Networks have found application in several fields within Computer Science, most notably Databases. The Semantic Network is a structure which uses pointers and pointer chains to represent the relationships between concepts. In a Semantic Network, a construct is an atom which is linked to its properties via edges. In addition, it is linked to other atoms to which it is related via "relationship links". An example of a Semantic Network is shown in Figure 1.

Figure 1
Semantic Network



Knowledge representation then, is essential to the construction of programs which exhibit "intelligent" behavior. Since knowledge representation is such a complex problem in its own right, it is prudent to ask what benefits are derived from constructing "intelligent" programs. To put it another way, why go to all the trouble? The justification for building "intelligent" systems as well as most of the mystery of AI itself can be found in the view that AI takes toward the fundamental relationship between man and machine. Traditional or mainstream Computer Science views the machine as the passive agent of the human. The machine is unquestioningly executing the instructions contained within an algorithmic program with no knowledge of or concern for their significance. The machine is simply responding to predefined and static direction. All "intelligence" (in the sense of sapient control) rests with the now absent programmer who coded the static list of instructions.

AI, on the other hand, takes a different view. The machine is considered to be an active partner which should cooperate with the user to accomplish the task at hand. The machine is capable of conceptual abstraction, autonomous learning, and self-direction. It is expected

that the machine will use its internal knowledge and these capabilities of abstraction, learning, and self-direction to further the user's objectives.

The contrast in the two views can be illustrated by an analogy. Consider the difference between training an animal (a dog, for instance) and teaching a human child. In general, the human child is presumed "intelligent" while the dog is not. In general, there are higher expectations of the human child than there are of the dog. While the dog may be expected to internalize a complex set of responses for specific environmental stimuli, the child is expected to conceptualize the behavior, to understand its motivation, and to abstract appropriate aspects of it for later use. In time, it is expected that the child will become self-directing and thus will depend entirely on these internalized abstractions to govern its behavior. The dog, on the other hand, will always require some vestige of the original stimulus used in conditioning in order to execute the appropriate behavior.

Traditional computing views the machine as the dog in the above example. The machine is expected to execute any algorithm, however complex, in response to

input from the programmer. The same input yields the same response time after time. There is never any expectation that the machine will acquire any experience which it will then use to improve upon the algorithm. Even those programs which appear to alter machine behavior (such as optimizing compilers) are merely executing static sets of instructions in response to specific input. These programs then, are algorithmic in character.

AI views the computer like the human child in the example given above. Actually, a more appropriate characterization of the AI view would be to say that AI considers the computer to be a trained assistant (much as a dentist views a dental hygienist). The dentist feels comfortable entrusting the task of cleaning teeth to the hygienist and having done so is free to pursue tasks requiring a higher level of expertise. It is from this view of the computer as a technical assistant that the entire subspecialty of Expert Systems arose.

Natural intelligence manifests itself in many ways. "Intelligent" beings understand natural language, they can abstract patterns, they can control behavior, they can learn, they can teach, and, they can play games. Artificial intelligence is concerned with

developing machines that can do all of these things. Thus, the field has many subdisciplines. Each subdiscipline is concerned with a specialized manifestation of intelligent behavior.

Natural Language Processing deals with developing machines that can communicate with their users in natural as opposed to artificial languages. Such machines would be much easier to use since the human is not required to learn an esoteric command set in order to interact with the machine. Much progress has been made in understanding written natural language text. Spoken language though, is another matter. The current state of the art faces a trade-off between vocabulary and speaker population. Machines with large vocabularies cannot understand a wide variety of speakers. Conversely, to be able to understand a wide variety of speakers, the machine is constrained to a limited vocabulary. AT&T Bell Laboratories is currently quite active in this area of research [5].

Pattern Recognition is concerned with developing computer systems that "see" in the same sense that a human or an animal does. Seeing is not accomplished with the eyes alone. It is the brain that interprets the signals transmitted by the optic nerve and assemblies

those signals into a recognizable image. Simple experiments with optical illusions indicate that the brain is an active participant in the process of visual perception. The eyes and optic nerve form the sensory system that reacts to light. The brain handles the task of interpreting the sensory data from the optic nerve. "Optical illusions" occur when the brain's interpretation algorithm is misled producing erroneous perceptions. Researchers in the field of pattern recognition are seeking interpretation algorithms which mimic those of the human brain [7]. These algorithms will, it is hoped, enable a computer system to perceive with the same accuracy as a human and then make decisions based on what it "sees". Unlike the human eye and ear, the sensory system supplying the input to the interpretation algorithm could perform in the ultrasonic, infrasonic, ultraviolet, infrared, electromagnetic, or any arbitrary region. These systems have applications in military intelligence and quality control. The military applications of pattern recognition are in the areas of tactical intelligence (friend/foe identification, passive surveillance, target acquisition, etc.) and guidance systems [6]. Commercial applications are in the areas of automated product inspection systems and automated analysis of plotted

data [7].

Autonomous Control and Robotics are two related fields which have attracted considerable attention. One reason for the current interest is the promise of sharply reduced manufacturing costs through the use of robotic assembly lines. Some believe that the savings realized from robotic assembly will revitalize the manufacturing industries of the Western industrialized nations, particularly the United States [10]. Autonomous vehicles are useful when conducting operations in areas hostile to humans, such as nuclear reactor cores, undersea operations, and work in space. Of course, numerically controlled machinery does not require meal breaks, rest periods, sick leave, vacations, or medical insurance; all of which increase labor costs. The current state of the robotics field is quite active with a number of large, well financed, corporations investing sizable amounts of capital. The field is hampered by several competing standards as well as some unsolved problems. More work needs to be done in the areas of end effectors (the "hands" of a robot), and feedback control systems (such as torque sensor control systems for assemblies using threaded components).

Gaming and Simulation Theory may be called the "grandfather" of Artificial Intelligence. The challenge of perfecting a computer program that could play games of reason (such as chess and checkers) was among the earliest tackled by AI researchers. Many "game" programs have been developed and the strategies for limiting large search spaces developed in the course of constructing those programs have found uses throughout the discipline of AI. Game programs remain a fertile field in their own right and have commercial value as entertainment products. In addition, simulation programs which model the behavior of physical systems (i.e. aircraft in flight) have grown out of the early work in decision spaces. Computer simulators have entertainment value as well, but they are more valuable as cheaper and safer alternatives to live training. Both the military and civilian aviation communities make heavy use of specialized hardware and software systems (called simulators) which are specifically designed to simulate the operation of particular aircraft.

Machine Learning and Computer Aided Instruction are two related fields which are both closely related to Expert Systems. All three areas are founded in Cognitive Science which is the study of how humans

assimilate and process knowledge. Machine Learning seeks to improve the performance of computer systems by allowing them to acquire new knowledge from the environment. This knowledge can be used to formulate heuristics, limit searches, or refine algorithms.

CHAPTER 3

OVERVIEW OF EXPERT SYSTEMS

Expert or Knowledge Base Systems are an important sub-specialty within AI. A principal reason for their importance is their enormous economic potential. That potential is so great that several companies, among them TechKnowledge Enterprises and Knowledge Engineering Enterprises [6], have recently been formed specifically for the purpose of bringing Expert System technology to the marketplace. In addition to the startups, several major corporations, including Texas Instruments and Bolt, Beranek, and Newman, have invested heavily in Expert System product development. Finally, the U.S. Government, particularly segments of the military have funded substantial research efforts in the area of Expert Systems [13].

The economic potential as well as the research and investment activity that it has fueled, can be attributed to characteristics unique to Expert System programs. The timing of the current interest in the technology is caused by the tremendous drop in the cost of computing power that has occurred in recent years. As the cost of computing power continues to drop, development and support of Expert System products

becomes cheaper. The current surge in popularity now enjoyed by Expert System products can be attributed to the fact that an economic threshold of affordability has just recently been crossed. Machines with sufficient power to support Expert Systems can now be purchased for under \$3000. Crossing this threshold means that a large enough customer base now exists to support the development of extensive Expert System product lines.

The terms Expert System and Knowledge Base System are synonymous [10]. For brevity this paper will use the term Expert System exclusively.

At the present time, the technology of Expert Systems is not sufficiently developed to make comprehensive generalities regarding structural details. At a high level though, some generalizations can be made [7]. From a high level view, a typical Expert System consists of the following parts:

1. The User Interface - Handles the communication with the system users. This portion is responsible for parsing the user requests and for displaying the conclusions reached by the system. In most cases, this component handles the explanation

Facilities which justify the system's conclusions to the user.

2. The Inference Engine - This component is responsible for control flow in the Expert System. Essentially this function consists of interpreting the information (facts) already known, using those facts to infer additional information, and tracking progress to the final or goal state. This component is sometimes called the inference mechanism or the rule interpreter.

3. The Knowledge Base - This component may have any of several different forms. It is the structure which stores the factual and procedural knowledge used by the inference engine.

While several methods for knowledge representation have been used in Expert System development, the most popular form is to represent the knowledge as a set of production rules. Expert Systems using this method are sometimes called Rule-Based Systems. The inference engine in a rule-based system then is responsible for tracking the system's progress toward the goal state and

for determining what action to take next. Usually this determination consists of selecting which rule to fire. Occasionally, as the deduction process proceeds, the inference engine will discover that it does not have enough facts to determine which of the available rules to examine next. It then calls on the user interface (sometimes called the Front End) to query the user for some missing piece of information.

User Front Ends in Expert Systems are similar to command interpreters found in operating systems but are somewhat more sophisticated. A major goal of Expert System development is to make specialized expertise available to the public at large. Consequently, an Expert System must be able to communicate effectively with a diverse user population. It cannot be presumed that the user population will be familiar with the specialized vocabulary and conventions of Computer Science. Further, the users cannot be expected to invest significant effort learning how to interact with the system. The user interface must accept the responsibility for communicating with the user at a high logical level using a convenient natural language vocabulary. Considerable advances have been made in the field of machine understanding of natural language text

since the early days of the Dartmouth Conference. Most Front Ends available today do a fair job of communicating in a natural language fashion; provided the discourse is confined to an appropriately restricted domain [5,7].

Frequently it is necessary for human experts to justify their conclusions to their clients. This requirement has been carried over to Expert Systems. Usually, the need for explanation or justification is met by a subsystem which presents the chain of deductions leading to a particular conclusion. This subsystem can be a part of either the user interface or the inference engine. As with the user dialog, the explanation facility must express itself using high level natural language constructs.

Some Expert Systems contain an additional component that permits the user to perform "what if" analysis. This capability is useful in cases where the user is attempting to learn the inference techniques captured in the Expert System or when the user is seeking guidance regarding specific observations to make. Further, the capability is helpful if it is necessary for the user to prioritize the observations to be taken and hence the user requires some knowledge

of the significance of a given item. "What if" subsystems are usually handled as extensions of the explanation subsystem.

Expert Systems have a long and rich history. The supporting technologies for Expert Systems are as old as the field of AI itself and have found application in all areas of that discipline. User Interfaces developed from the early work in natural language understanding and synthesis. Inference Engines are products of simulation and gaming theory which developed efficient searching programs. Knowledge Bases are the direct result of work in knowledge representation.

The first successful Expert System was a program called DENDRAL [7], which was developed as a cooperative project between the Chemistry and Computer Science departments of Stanford University in the early 1970s. DENDRAL identified organic compounds by comparing the mass spectrometer traces of the sample with a library of mass spectrometer traces for standard compounds and elements. From a Computer Science point of view, DENDRAL was a significant advance as it demonstrated the feasibility of modeling the human reasoning process within a computer program.

The next significant development in Expert Systems was the program MYCIN [6,10]. This program was developed at the University of Pittsburgh in 1976. Its purpose was to identify certain bacteriological infections and recommend courses of treatment. MYCIN featured a pseudo-natural language Front End, explanation facilities, and an advanced inference engine. The inference engine was significant in two respects; it incorporated fuzzy reasoning which allows it to deal with inexact or incomplete data, and it could be decoupled from the knowledge base. The ability to handle inexact or incomplete data was a major step forward in Expert System development because it greatly expanded the class of problems which a particular rule base could address. Being able to decouple the inference engine meant that it was no longer necessary to develop custom inference engines for each knowledge base. By eliminating this effort from Expert System development, the development process was greatly simplified.

Expert Systems demonstrated their commercial potential in the program R1 [7]. R1 was developed by a partnership between Digital Equipment Corporation and Carnegie-Mellon University. The system had immediate

commercial application. In fact, it was developed specifically to meet Digital Equipment's need to make computer configuration expertise more widely available. The problem was that customers were purchasing VAX equipment from Digital (through the company's sales force) but were not always buying all the necessary or appropriate equipment to meet their requirements. Digital found itself swamped with requests to rectify after-sale configuration problems at their own expense. Obviously, these requests presented Digital with a significant cash flow problem as well as a puzzling dilemma. Should the company rectify the problem and take the loss or charge the customer extra and lose his goodwill? It was clear that the best solution was to avoid the problem in the first place by insuring that the system as ordered would satisfy customer expectations. Unfortunately, the VAX product line was new at the time and specialists in VAX configuration were rare. Digital took the innovative approach of teaming with a recognized leader in AI research (Carnegie-Mellon) to develop a computer program which captured the available VAX configuration expertise. The program, R1, went through several modifications and was eventually renamed XCON. According to Digital's own estimates, use of XCON has saved the company over \$20

million annually by reducing the volume and complexity of after-sale configuration adjustments.

In fact, the success of XCON inspired Digital to fund the development of an similar product, XSEL. This product captures VAX product expertise in order to assist sales personnel in the promotion of the VAX product line.

An important threshold has been crossed by the expert system DIPMETER ADVISOR [6,7], developed by Schiumberger-Doil Research. The program locates economically promising mineral deposits by examining borehole strip recordings. Recently, DIPMETER ADVISOR located a rich and lucrative mineral deposit that had escaped detection by human analysts. By doing this, DIPMETER ADVISOR validated a long standing claim of AI researchers that someday Expert Systems would accomplish tasks that humans would not accomplish acting on their own. Validating this claim is an important event in the history of Expert Systems because their value has now been established beyond question.

CHAPTER 4

THE INVESTIGATION - INITIAL ACTIVITY

The investigation proceeded in stages. The first phase consisted of selecting a representative test problem typical of those likely to be encountered in a commercial technical environment. The next phase was the selection of an appropriate development shell with which to implement the demonstration Expert System. This chapter discusses these two activities. The next chapter will discuss development and testing of the demonstration Expert System.

Selection of the Test Problem

The problem which I selected for implementation was that of predicting short-term or short-range weather patterns using readily obtained local observations. The problem meets the requisite criteria.

It is neither trivial nor contrived. While both numeric and simple decision tree programs exist for weather prediction, neither is well suited to the problem. The decision tree approach yields predictions that are too general and thus are of little value. The

numerical approach has to date not been applicable to localized phenomena. The dynamics of the atmosphere are decidedly non-linear and thus are described by non-linear partial differential equations [9]. Closed form solutions to such equations are beyond the scope of contemporary mathematics. The numerical prediction models then, simplify the problem so that the processes can be described with linear partial differential equations, and thus can be solved by numerical approximation techniques. By simplifying the problem though, the models lose much of their power to predict small scale (in meteorology the term is microscale) events [3].

There exists a well established set of heuristics that can be brought to bear on the problem. As early as the 16th Century, the American scientist and statesman Benjamin Franklin recognized that the weather in Philadelphia was very often the same as the weather in New York City had been just a few days earlier [3]. Franklin had recognized that local weather was related to travelling atmospheric systems. Shortly after World War I, a group of distinguished Norwegian meteorologists at the University of Bergen, Norway, developed a model of a low pressure system that has become recognized as a

classic [1]. This model, known as the "Norwegian Wave Cyclone Model", forms the basis of modern weather forecasting methods (including the one I am using).

The particular inference technique chosen for this expert system was developed by Alan Watts, a fellow of the British Meteorological Society and a professional weather forecaster. The model views the atmosphere as a collection of distinct, homogeneous "air masses" and explains weather behavior in terms of "interactions" between these "air masses". Air masses are characterized as either HIGHS or LOWS depending on the magnitude of the air pressure in the particular air mass relative to that in its neighbors. The magnitude of the pressure difference between two adjacent air masses (sometimes called "systems") determines a quantity known as "system strength". System strength is never measured directly and is completely non-deterministic. The model uses the device of "system strength" to handle the non-deterministic and inexact aspects of the local weather forecasting problem.

The Watts model is documented in two reference sources [14,15] which are quite technical in nature and require an appreciation of meteorology to use effectively. Watts himself [15] points out that the

techniques are best learned by experience. Further, since Watts is British, the techniques are explained and illustrated using weather patterns common to the British Isles rather than to North America. The training investment to learn the method directly then is quite substantial. It is, therefore, more cost-effective to encapsulate the method in an expert system than to master it directly through training.

The problem of meteorology has an additional characteristic which makes it uniquely suited to Expert System research. Its heuristic set is easily extended. While the Watts model provides a rich set of general heuristics, localized phenomena (such as mountains, lakes, urban heat islands and the like) have a significant effect on the prevailing local weather. It is possible to develop heuristics which account for these local effects. Thus, the demonstration system could itself be used as a research tool to explore heuristic development.

Selection of the Development Shell

Once the test problem was chosen, the focus of the

investigation shifted to selecting an appropriate development shell. Major factors in shell selection were cost and availability. Other important considerations included the hardware on which the shell ran, its representation structure, and its ability to accept extension.

As previously stated in Chapter 1, one of the goals of this project was to select a shell that would run on an MS-DOS compatible machine. The shell had to be available to the University, as well as being available at my home location. Further, since meteorology is a numeric intensive science, the shell's representation structure had to accomodate numeric or mathematical processing. Of course, the shell also had to possess an appropriate user interface including high level discourse and explanation facilities.

The shells available to the University were reviewed and the choice was narrowed to two; Personal Consultant Plus from Texas Instruments and MICROEXPERT from McGraw-Hill. Both shells met the basic requirements discussed above but MICROEXPERT was more generally available, required less specialized hardware, and cost significantly less. MICROEXPERT was therefore selected as the development shell.

MICROEXPERT has many characteristics which make it attractive to commercial engineering firms. It is generally available through McGraw-Hill's distribution network, it runs on a plain-vanilla MS-DOS machine (Intel 8088 design), and requires only 512 (K) of user available memory. The source code is supplied with the product and is written in Turbo Pascal. Since Turbo Pascal is a generally available development system and limited permission to modify is given in the MICROEXPERT licensing agreement, the user is free to tailor MICROEXPERT to their own needs. These factors make MICROEXPERT the type of product needed to encourage the growth of microcomputer-based Expert Systems.

Description of MICROEXPERT

MICROEXPERT is supplied on a PC-compatible 5.25 inch floppy disk which contains source files, sample knowledge base files, and executable files. Example consultations may be conducted using the sample knowledge bases supplied. An off-line utility program, CROSSREF, is available to examine the knowledge base files independent of a consultation. Source code is supplied for both MICROEXPERT and CROSSREF. The

knowledge base files are stand-alone ASCII files which must be constructed with a user-supplied text editor capable of ASCII output.

The high level logical structure of MICROEXPERT is straightforward. Before beginning a discussion of that structure, I must point out that limitations of Turbo Pascal and the PC machine design require that the program's physical structure be much different than its logical one. The physical implementation structure of MICROEXPERT will be discussed in Chapter 6 which deals with the extensibility analysis. For the moment, I will focus attention on the program's logical structure.

The program is logically partitioned into three major components, the user interface, the inference engine, and the supporting utility routines. The user interface is responsible for the system's interaction with the outside world (both the human operator and the MS-DOS operating system). The inference engine traverses the rule base and performs the deductive reasoning. The utility routines handle low level support tasks such as file and string manipulation.

A consultation begins when the user invokes MICROEXPERT from the DOS command interpreter. Provision

has been made for the user to supply the name of the knowledge base file as an argument in the DOS command line. The user interface checks that command argument first and prompts the user for the file name only if the argument is empty. If the user responds to the prompt with a null line (carriage return) the user interface calls a custom implementation of the "dir" procedure which searches the DOS default directory for knowledge base files. MICROEXPERT requires that knowledge base files be given the extension <.kb>. If the user does not supply that extension to the file name typed in, the program automatically appends it to the entry before beginning the search. Knowledge base files are presumed to exist. Entering a non-existent file name causes the user interface to return to the file name prompt. When the user interface locates the specified knowledge base file, it then begins parsing the file and forming the various lists required by the inference engine (details of the rule base and inference engine will be discussed below). Once the rule base has been read, the consultation begins.

The consultation proceeds like a directed interview. The system first asks the user for a goal. When a goal is entered the inference engine begins

traversing the rule base, looking for a way to reach that goal. The inference engine and user interface cooperate throughout the consultation with the inference engine traversing the rule base and making requests for information, while the user interface constructs questions for the user in order to satisfy those requests. At any point in the consultation, the user may query the system to find out what it has determined so far, the current subgoal on which it is working, and the consequences of a given answer (a "what if" analysis). The user interface is responsible for handling all those situations.

The logical structure of the user interface is similar to that of a command interpreter but more complex. When the user interface constructs the rule lists, it assembles a list of valid answers for each rule. These answer lists are compared against user input to determine if the user input is legal. User queries to the system are handled by reserving certain character strings as command responses. When a reserved character string is encountered, the user interface vectors control to a specific procedure which services that command. For example, if the user entered the reserved word WHAT, the user interface would call the

procedure WHAT which prints out a list of the facts that the system has deduced thus far in this consultation. The command WHAT is interpreted as "What facts have been determined thus far?". The reserved command strings are:

WHY

WHAT

HOW

WHATIF

QUIT

The interpretations of each command will be described later in the discussion of the inference engine.

MICROEXPERT is designed to function efficiently. One of the compromises made to achieve efficiency was to limit the predicates used in the production rules to one, the predicate "is". Further, the conditions and conclusions in the rule base must have names no longer than 19 characters. These limitations would have severely restricted the domains for which the system could be used were it not for the ingenious design of

the user interface. The user interface supports "prompts" and "translations". These constructs are arbitrary character strings of unlimited length which the knowledge base developer may supply. Prompts are questions which request a response from the user. Translations are statements of fact that are output in lieu of the conclusion name itself.

For example, consider the following rule:

if A_nose_cold is yes, then A is dog.

Without prompts or translations, the user interface would ask

A_nose_cold?

If the user responded with the value "yes" then the system would conclude:

A_nose_cold is yes; A is dog

The output is understandable but not very elegant.

With prompts however, the question requesting a value for A_nose_cold becomes "Is animal A's nose cold?". The output becomes "Animal A's nose is cold; Animal A is a dog." Prompts and translations are the responsibility of the user interface.

The inference engine in MICROEXPERT is a simple implementation of a classic backward chaining inference mechanism. Before discussing the mechanism itself, it is necessary to understand MICROEXPERT'S internal rule base representation.

MICROEXPERT uses a decoupled knowledge base. A decoupled knowledge base means that the domain of the system is not fixed. To change the domain, one merely has to replace one knowledge base with another. In order to achieve this flexibility, the knowledge representation structure must be constant. MICROEXPERT requires that its knowledge base be represented as production rules. The rules have the form [condition]!then![conclusion]. Conditions are "if" clauses (such as "if A_nose_cold is yes") and conclusions are unqualified clauses (i.e. "A is dog"). All clauses must be of the form {attribute}!is!{value}. Recall that attributes and values are limited to 19 characters each and that "is" is the only allowable

predicate. Attributes are analogous to variables in numeric expressions in that they can "take on" values.

Internally, the rule base is represented in a linked list data structure constructed and populated by the user interface. This structure associates rules with their clauses and attributes. Attributes are also linked with their values as well as with their prompts and translations. A sample entry from the rule table structure is shown in Figure 2. The internal representation of prompts and translations is shown in Figure 3.

The inference engine uses the rule table structure to keep track of the rules and their associated attribute and value pairs. The inference engine considers any conclusion of any rule to be a goal candidate and maintains a list of the conclusions in an "available goals" list. It also maintains other data structures. The "goal stack" contains the goals on which the system is currently working, and the "context stack" contains a list of the facts which the system has deduced or been supplied with thusfar. To support explanation, the context list also contains pointers to the rules that determined the facts on the list (including a special pointer for user input).

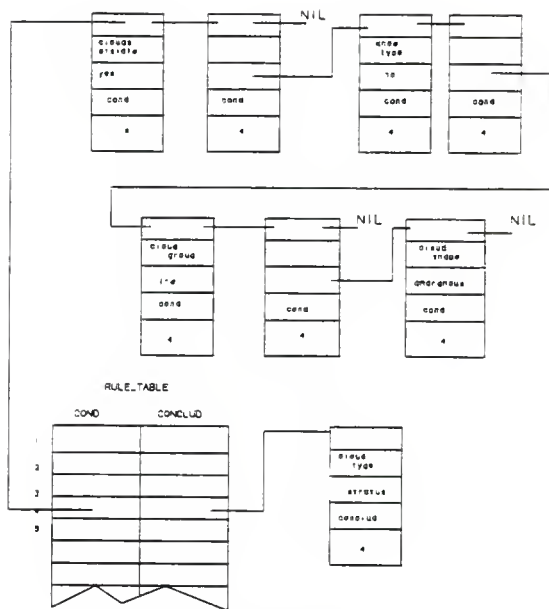


Figure 2
Internal Representation
of a Rule

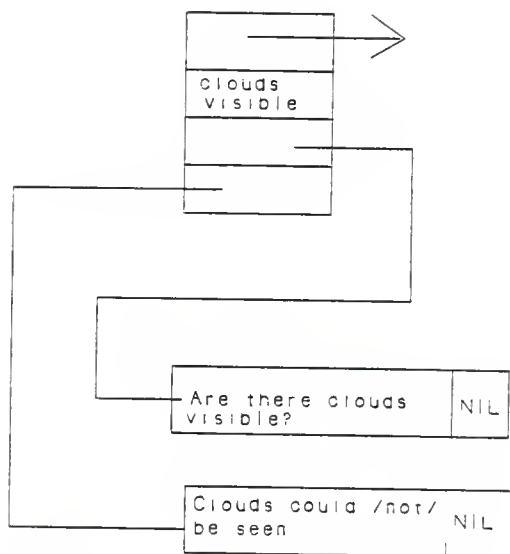


Figure 3
Internal Representation
of
Prompts & Translations

When the user supplies a main goal, the inference engine scans the rule table for any rule containing that goal in its conclusion. Upon encountering such a rule, the inference engine then checks the context list to see if the rule's conditions are satisfied (TRUE). Rules can have multiple conditions (joined by the word AND). If all the conditions are satisfied, the conclusion is placed on the context list and the consultation is finished. If all the conditions are not satisfied, the first unsatisfied (unknown) condition is placed on the goal stack and the inference engine scans the rule base with that attribute as the new goal. Goals can be "stacked" to several levels (the maximum number is implementation dependent). If a value for an attribute is needed but no rules are available to determine it, the attribute is placed on the goal stack and the user interface is called to prompt the user (using the supplied prompt if there is one) for the value. As each goal is determined, the goal stack is popped and the value determined is placed on the context stack. When the goal stack is empty, a value has been determined for the main goal and the consultation is over.

The functions of the command strings can now be explained. The "WHY" command is interpreted as "Why is

this question being asked?". The current rule being evaluated is printed out as an answer. The "WHAT" command is interpreted as "What has been determined so far?" and causes the system to display the context list. The "HOW" command requires a conclusion number (the position in the context list) as an argument and displays the rules which can be used to determine that conclusion. The "RULE" command accepts a rule number as an argument and displays that rule. The "WHATIF" command accepts an attribute value as an argument and displays a list of rules that would be evaluated if that value were placed on the context list. The "QUIT" command causes the consultation to be terminated immediately and causes the current context list to be displayed.

Before concluding our discussion of MICROEXPERT, mention must be made of some unique features. MICROEXPERT allows attributes in both the condition and conclusion side of rules to be user-written functions or procedures. An example of such a rule is <if function mean (a) is 22 then result is procedure printit (results.dat)>. When such rules are encountered, the inference engine strips off the "function" or "procedure" header and calls the user supplied module.

Actual parameters are passed to the called modules in the global structure "parm_array". MICROEXPERT uses this mechanism itself to handle numeric processing. It is equipped with a procedure "math" for numeric operations and a function "compare" for logical arithmetic operations. This feature was useful in the development of the demonstration Expert System. It will be revisited in greater depth in Chapter 5.

This chapter has discussed the selection of the meteorological test problem and the development shell MICROEXPERT. The next chapter will address the development and testing of the demonstration Expert System, beginning with the inference techniques to be captured and ending with the final development tests.

CHAPTER 3

SYSTEM DEVELOPMENT AND TESTING

This chapter discusses the development and testing of the demonstration Expert System. Development of this system is the second phase of my investigation into microcomputer hosted Expert Systems. The final phase of my investigation, the MICROEXPERT extensibility analysis, will be discussed in the next chapter.

Capturing Inference Techniques

Development of an Expert System begins with capturing the expertise to be represented in the system. Frequently, capturing this expertise or these inference techniques requires that the system developer (also known as a knowledge engineer) seek out and interview recognized experts in the domain. From these interviews, the knowledge engineer constructs a consistent representation of domain knowledge and inference rules (heuristics).

Constructing a consistent representation is not often a simple task. Experts frequently disagree on the structure, correctness and priority of inference rules. Further, it is often the case that the expert cannot

explain the basis for his or her decisions. Since heuristics are often learned by experience, experts tend to rely on "hunches" or "intuition" or "gut feeling" rather than on a formalized set of inference rules. Many times rule development is a stepwise refinement process of drawing up rules and then having the experts review them for correctness.

The Watts Inference System

Fortunately, capturing the expertise for the demonstration system did not involve such a complex process. Alan Watts, a British meteorologist, had formulated a set of inference rules which predict near-term, immediate vicinity weather from visual observations of cloud structure, temperature changes and wind direction. To understand Watts' rules, it is necessary to appreciate the connection between atmospheric physics and clouds.

Clouds are composed of condensed water vapor. They form when moist air is rapidly cooled. At the cooler temperature, the air cannot hold all of its water vapor in suspension. The water vapor condenses into microscopic droplets which are visible as a white mist (what is sometimes called fog).

In the atmosphere, air temperature decreases with altitude. When moist air is forced to rise, it cools rapidly and clouds form. The structure of the cloud will vary depending on what kind of circulation mechanism is causing the air to rise. By examining the structure of the cloud formations, experts can often tell what circulation patterns are present and thus predict the weather [1].

Circulation patterns in the atmosphere are the product of complex turbulent flows triggered by solar heating [9]. While these flows cannot be modeled deterministically and the equations governing them cannot be solved in closed form, some helpful generalizations can be made. A common generalization is to divide the atmosphere into two types of flow regimes, the upper level flow and the lower level "weather systems". This approach is a simplification and thus does not fully account for atmospheric dynamics, but it has been used for years to develop useful weather forecasts.

The upper level flows in the model are sometimes referred to as steering currents or jet streams. These are high speed, high altitude, and stable circulation systems which encircle the earth at various latitudes. The weather in Kansas is most influenced by the "mid

latitude" jet which covers most of United States [9].

Weather systems are active at lower altitudes and travel from place to place. They are said to "ride" the jet streams in that the jets push them along and direct their motion. It is these weather systems that are shown as the "highs" and "lows" on the nightly weather map. In meteorology, a "high" is said to have "anticyclonic" circulation and is called an anticyclone whereas a "low" has "cyclonic" circulation and is called a cyclone.

Although the term "cyclone" has become associated with tornadoes in the popular mind, it refers to the circulation pattern about any region of low pressure, not just the intense low pressure at the center of a tornado. This circulation pattern is the result of the Coriolis force acting on the winds flowing into the center of low pressure. In the northern hemisphere, the Coriolis force causes these winds to spin in a clockwise direction (in the Southern hemisphere, the Coriolis force acts in the opposite direction). Conversely, winds flowing out from a center of high pressure are spun in a counter-clockwise direction by the Coriolis force.

A "high" is a region of subsiding (sinking) air which is characterized by fair weather. Since the air pressure is higher in the center of the system, airflow is directed downward (subsidence) and outward from the system center. The sinking air in a high is the reason for the fair weather.

A "low", on the other hand, is a region of rising air which is characterized by unstable (usually undesirable) weather. The lower pressure at the system center causes air to flow inward and upward, contributing to the formation of clouds and precipitation.

Shortly after World War I, a group of meteorology researchers at the University of Norway in Bergen, developed a model to describe the formation, structure, and life cycle of a cyclonic system. As described by the model, cyclones are circulation systems that resemble waves on the ocean. The cyclones build, break (crest) and then dissipate. Because of this similarity to water waves, the model is called the "Norwegian Wave Cyclone Model".

Most low pressure systems can be accurately described by the Norwegian Wave Cyclone Model. High

pressure systems are the regions of subsidence (sinking) airflow that exist between the cyclones to maintain equilibrium. The model (and subsequent observations) enable meteorologists to chart the cloud formations typical of Norwegian Wave cyclones.

Figure 4 [15] shows a sketch of the cloud structures of a typical Norwegian Wave Cyclone. Watts' inference technique is based on this model. It attempts to fix the position of an observer relative to a Norwegian Wave cyclone. For example, assume the cyclone in Figure 4 is tracking from left to right on the page, an observer would first notice the high cirrus clouds as the cyclone is approaching. The observer would detect the thickening cloud mass as the cyclone neared and finally would notice the stratocumulus and cumulonimbus clouds in the center of the system. As the system receded, the cumulonimbus clouds would give way to stratocumulus and finally to fair weather cumulus as the following high approached.

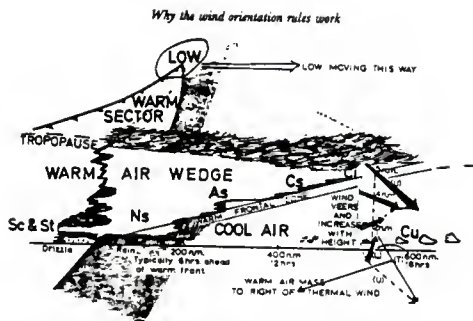


FIG. 4.1 How winds veer with height ahead of a warm front or occlusion. The upper cloud moves with the upper wind (U) so that the thermal wind (T) may flow to keep Low temperature on its left. Heights and distances are typical for vigorous (one) warm front.

Figure 4

Typical Cloud Structure

of a

Norwegian Wave Cyclone

Unfortunately, applying Watts' inference scheme to the real atmosphere is not as simple as the preceding explanation may suggest. Cyclones and anticyclones are not completely decoupled from the upper level jets. Sometimes the high speed winds of the jets distort the cloud structure, causing it to differ from that predicted by the classic model. Also, cyclones do not often form in isolation but rather exist in close groups (called "families"). The cloud structures of family members often intermingle, requiring the observer to sort out which clouds belong to which cyclone. Further, cyclones in various stages of development often have incomplete cloud structures. It is even possible for members of the same family to be in different stages of development simultaneously.

These are just some of the factors which make the weather prediction problem non-deterministic. Watts took two approaches to handling this non-determinism in his inference technique. In his earlier book [15], he chose to describe the weather forecasting problem in technical terms. He developed several sets of rules that required various types of meteorological measurements (mostly wind velocity at various heights). When one possessed these measurements, one could use the

heuristics Watts supplies to make reasonable weather predictions.

Unfortunately, such measurements are not normally available to the average individual. Even if the average person were to have access to the detailed meteorological data required, substantial expertise is required just to prepare the data for submission to an Expert System. In his more recent book [14], Watts takes the other approach. The book is a collection of sky photographs which are annotated with inference rules and explanations. The user matches the sky observed with that of the photograph, applies the associated inference rule, and formulates a prediction. This approach has the advantage of not requiring detailed meteorological data, but the disadvantage that the templates may not exactly match. The user must accept the responsibility for finding the closest match to an available template.

***Capturing the Watts Technique
for the MICROEXPERT Rule Base***

Clearly, the second approach Watts used [14] was the more appropriate for an Expert System of the type I

was developing. Not only would the projected rule base be smaller, but the system would be easier to use and test since the input required would not need extensive preconditioning. Unfortunately, MICROEXPERT does not accept graphic input so direct template matching was not possible.

Watts addressed that problem by providing a table of cloud classification in [14]. Using this table, I was able to construct a decision tree which classifies clouds based on their height and structure. This tree is shown in Figure 5. Because of MICROEXPERT'S 19 character limitation on attribute names, the resulting rules contain some less than obvious values for cloud structure description. This was a fundamental limitation. I could not unambiguously classify a cloud subtype (i.e. stratocumulus is a subtype of the base type cumulus) without resorting to long value names. The value length limitation forced me to compress the names which made them less understandable. I was compelled to extend MICROEXPERT in order to solve this problem. The extension I developed permits a user to request definitions for rule responses that are not clear. Details of this extension will be discussed later in the chapter.

Figure 5
Meteo.kb Decision Tree
(Plate I)

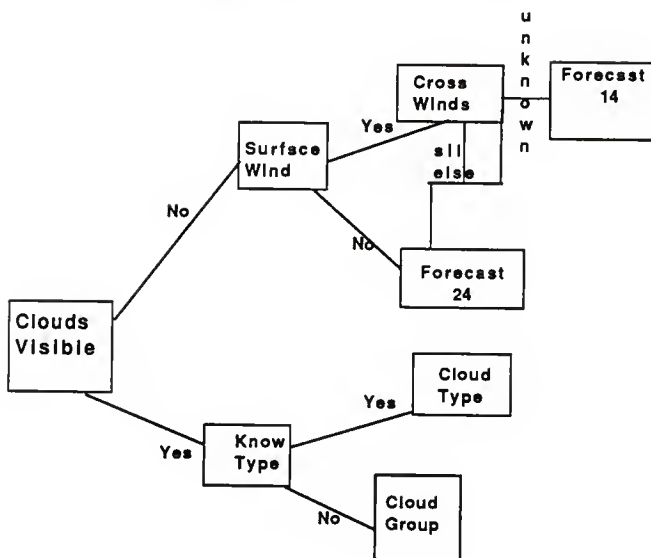


Figure 5
Meteo.kb Decision Tree
(Plate II)

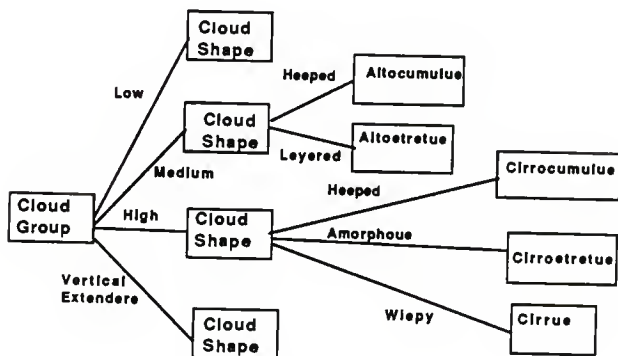
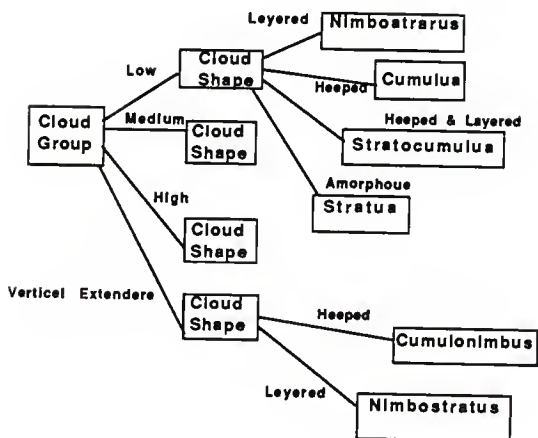


Figure 5
Meteo.kb Decision Tree
(Plate III)

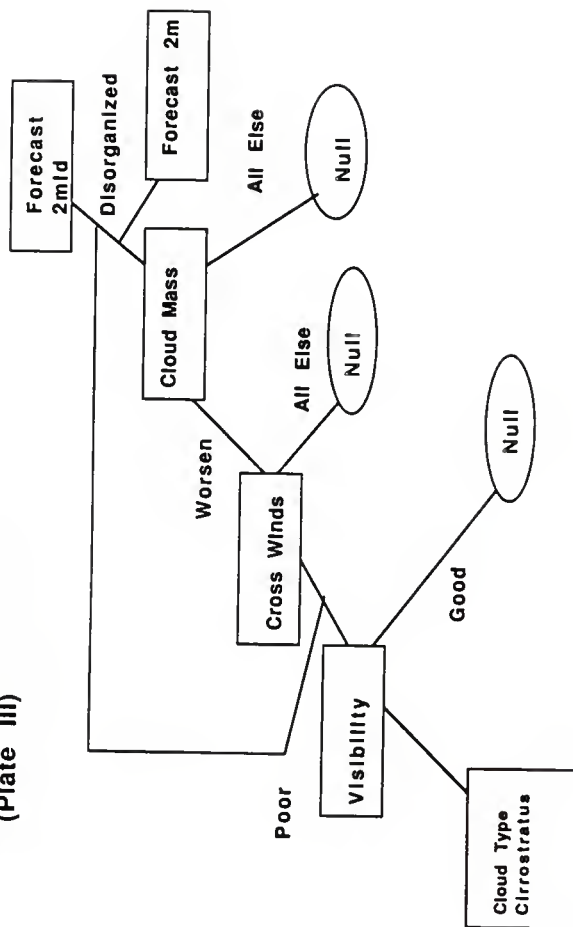


Figure 5
Meteo.kb Decision Tree
(Plate IV)

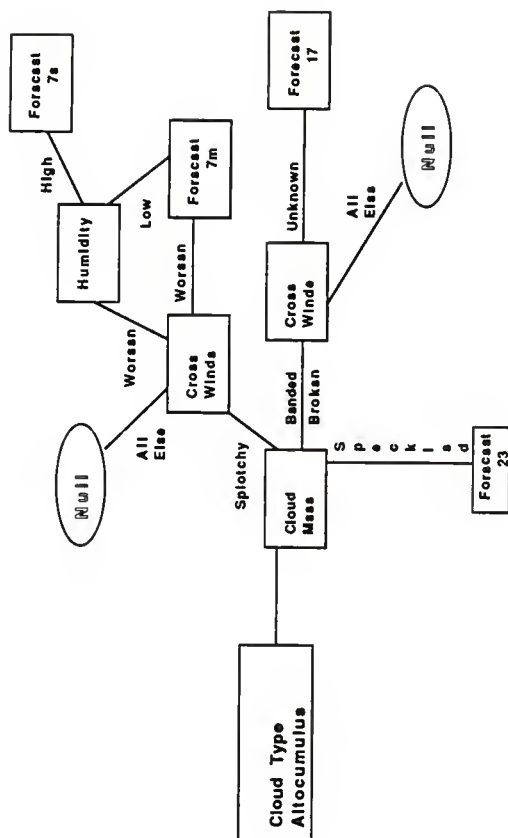


Figure 5
Meteo.kb Decision Tree
(Plate V)

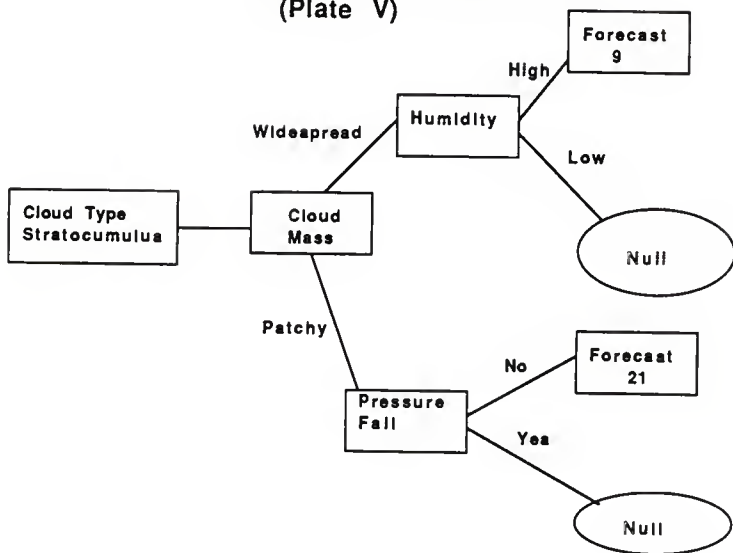


Figure 5
Meteo.kb Decision Tree
(Plate VI)

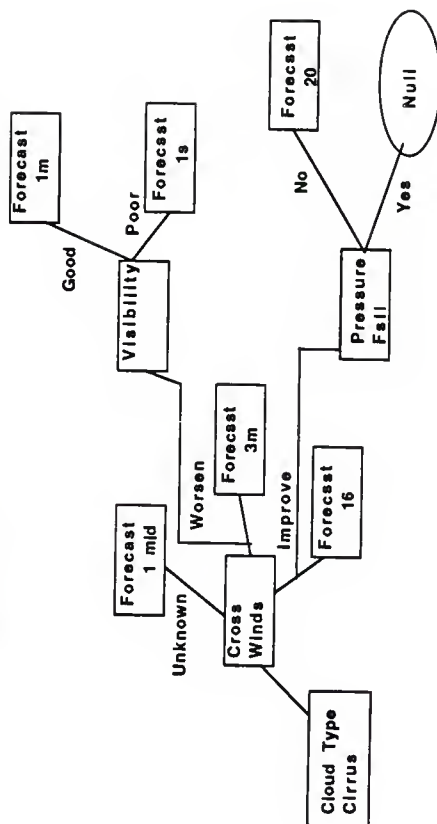


Figure 5
 Meteo.kb Decision Tree
 (Plate VII)

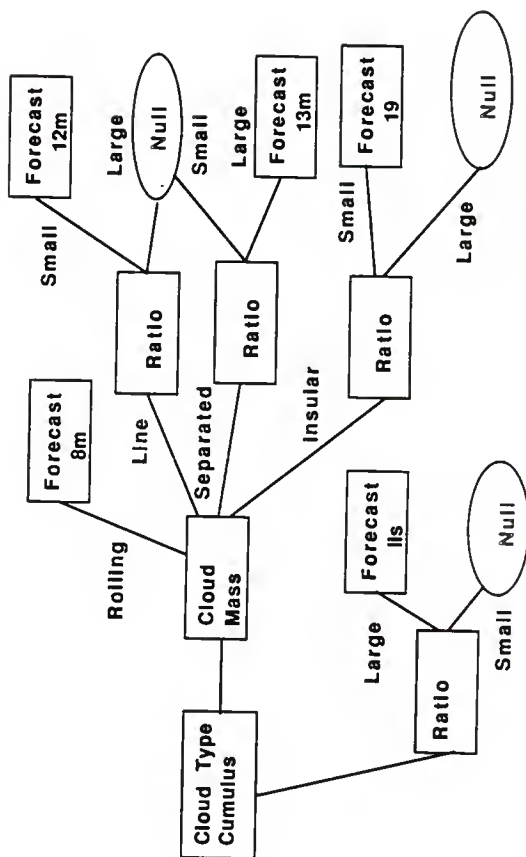


Figure 5
Meteo.kb Decision Tree
(Plate VIII)

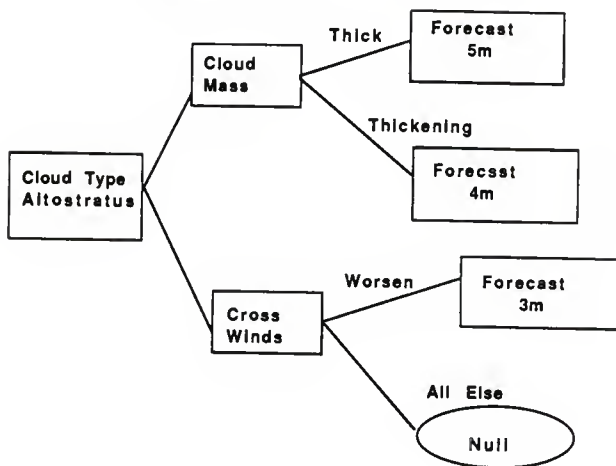


Figure 5
Meteo.kb Decision Tree
(Plate IX)

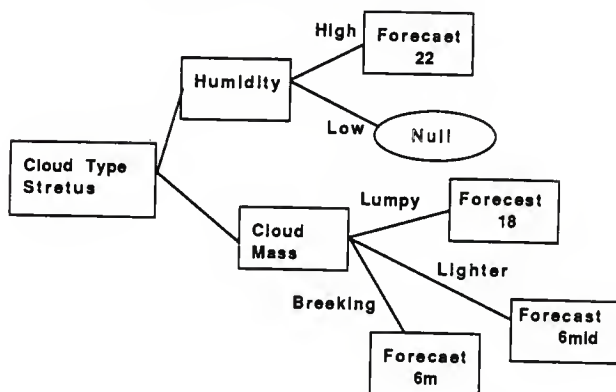


Figure 5
Meteo.kb Decision Tree
(Plate X)

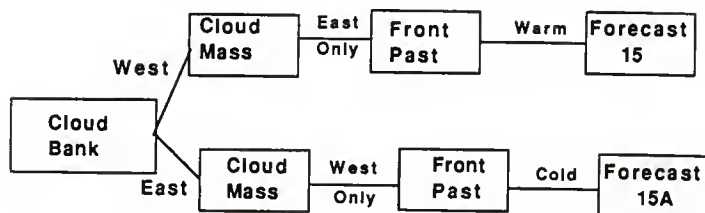


Figure 5
Meteo.kb Decision Tree
(Plate XI)

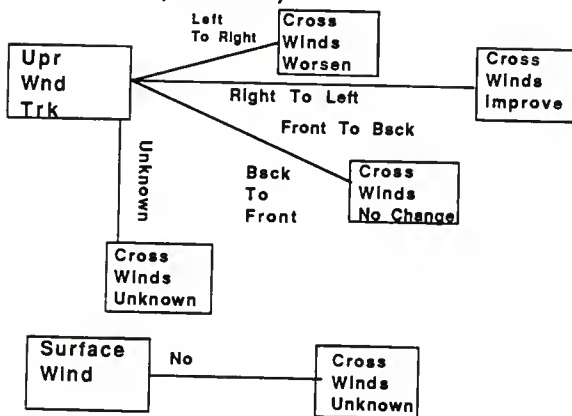
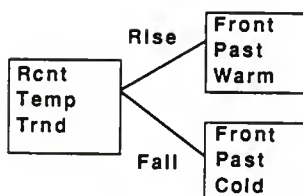


Figure 5
Meteo.kb Decision Tree
(Plate XII)



The cloud classification problem formed my prototype rule base. That rule base contained 11 rules when completed. In addition to developing the rules themselves, the prototype allowed me to develop an algorithm for ordering the rules in the rule base. The algorithm is based on the fact that MICROEXPERT uses simple backward chaining to traverse the rule base. In simple backward chaining, the rules containing the current goal state are traversed in numerically increasing order. If the first (lowest numbered) rule fails, the next highest is attempted. The process is completed until all rules containing the goal in their conclusion side are exhausted.

My ordering algorithm takes advantage of the monotonic behavior described above. The algorithm proceeds as follows:

- 1) Develop all the inference rules
(decision trees are helpful at this step).
- 2) Group the rules by their conclusion sides such that rules containing common goals are collected into sets.

- 3) Order the rules within each set in decreasing order by the number of clauses on their condition sides.

It should be noted that the above algorithm was developed for the special case in which the conclusion side of a rule contains only a single clause. In the more general case where the conclusion is a string of compound clauses, an additional step would be required between steps 2 and 3. Let that step be called 2a. It states: Order the rule sets in decreasing order by the number of clauses on their conclusion side.

This algorithm saves quite a bit of time in ordering the rules of the rule base. The goal is to force the inference engine to address the most complex cases first. These cases force the inference engine to deduce the most facts, which encourages the processing of subgoals early. The net effect is that the system requests data from the user more efficiently. As it moves on to the more simple cases there is a monotonically increasing probability that the data it requires to evaluate the next rule is already on the context list. Having the data on the context list eliminates the need to request it from the user and

allows the system to process the remaining rules even faster. Thus, the system will converge toward a solution more rapidly.

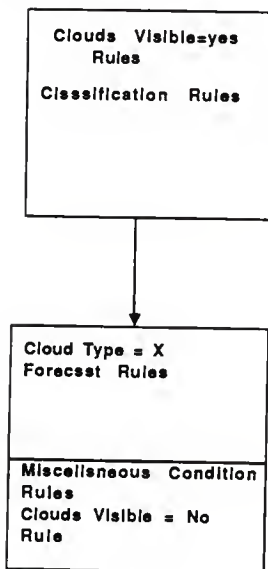
In the case of MICROEXPERT however, the algorithm contains a flaw. Since MICROEXPERT uses simple backward chaining, the rule base must contain guard rules which partition the common goal rule sets (step 2). The function of guard rules is best illustrated by an example from an early version of the full rule base.

A structural diagram of that rule base is illustrated in Figure 6. Note the ordering of the rules. The order is based strictly on the number and not the content of the condition clauses. The problem arises because the first group of "forecast" rules all have cloud_type in their conditions. Cloud_type is presumed to be determined in the first group of rules. Note that the cloud_type rules all contain the condition "clouds_visible is yes". There is a "forecast" rule that handles the "clouds_visible is no" condition but it is reachable only if the preceding rules fail.

Figure 6
Structural Diagram
of
Flawed Rule Base

Are There Clouds?

Clouds Visible



Forecast

The difficulty should now be apparent. Suppose the user selects "forecast" as the main goal. The system begins to process the first "forecast" rule and identifies cloud_type as a subgoal. The user enters "no" for the clouds_visible value. All the cloud classification rules fail. The system then prompts the user for cloud_type, saying "What type of clouds are they?". Presumably the system is referring to the clouds that cannot be seen and thus do not exist. The system is in deadlock. It cannot proceed until it has a value for cloud_type. It cannot obtain any value for cloud_type because there are no clouds.

The ordering algorithm created the problem when it ignored the significance of the forecast "guard rule" that handles the "clouds_visible is no" case. That rule should be placed first in the "forecast" rule section. In that position it would fail if there were clouds and allow the cloud based forecast rules to be processed, but end the consultation gracefully if no clouds were visible.

MICROEXPERT has another implementation characteristic that impacts rule base design. The user interface constructs a list of valid answers for each rule. Except for the reserved commands, an answer is

valid if and only if it is contained in a clause on the condition side of a rule. Thus, all reasonable responses to a prompt must be encoded in a conditional clause in order to be accepted. This is required even if no rule is available to handle the value. The system presumes that its rule base is complete. If it has no rule to process a response, that response is not legitimate. In other words, the system does not know that it is not omniscient in its domain. Instead, it presumes that the user has made an error. The net effect of this implementation is to require more rules. Rules that conclude "I don't know what to do" must be included in the rule base for every case in which a reasonable response cannot be processed.

System Output - The Forecast Tables

The output of Watts' inference technique consists of brief explanatory statements and forecasts for the next few minutes to several hours depending on the specific forecast. The forecasts predict precipitation, wind and sky conditions, temperature trends and barometric pressure trends. They also give the time scale over which the predicted changes are likely to

occur. Since MICROEXPERT limits attributes to 19 characters, I could not output the descriptions directly. In my original concept, I intended to construct static tables from the Watts descriptions [14]. The Expert System would produce pointers (e.g. fcst_1_mid) which would refer the user to the entry in the static table.

MICROEXPERT'S ability to incorporate procedure calls as the conclusion of rules provided for a more elegant approach to output design. Instead of the system just generating the reference pointer, it is possible to cause it to supply the pointer to an output procedure and then invoke that procedure. The output procedure would access the default directory for the text file indexed by the pointer (an 8 character file name) and display the text file on the screen. This is the second extension I was forced to make to MICROEXPERT. A detailed explanation of both extensions will be given in later in the chapter.

At this point in the investigation, the implementation of the demonstration system was now complete. The system was ready for testing.

Testing the Demonstration System

Testing an Expert System presented an unusual challenge. The obvious approach was to test the system's by developing "test vector" consultations whose conclusions were known in advance. The challenge was to generate a set of test vectors which spanned the domain.

The solution I selected was to view the Expert System as an algorithmic program containing branches. Each rule in the rule base was produced from a decision tree. When these trees were interconnected, they formed a set of branching paths from input to output. It was possible to work backward from the outputs and reconstruct a unique branching path for each one. Although the effort required was tedious, it was finite.

Once the paths were constructed, it was a trivial matter to run the consultations needed to produce the results. It should be noted that testing the Expert System in this way is analogous to CO test coverage in an algorithmic program. Every path through the system is exercised at least once.

In the case of the demonstration system, displaying the list of acceptable inputs for each prompt (question) proved useful. This procedure is a shorthand way of looking at the system from the input side. It was valuable as an analysis tool for this system because of the omniscient presumption inherent in MICROEXPERT. Specifically, the test disclosed that "low" was not an acceptable value for HUMIDITY when clearly the humidity is often low. That situation occurred because the rule base did not have a rule which inferred anything from low humidity. In order to make the input acceptable, a rule requiring the value "low" as a condition was added to the rule base.

The Extensions to MICROEXPERT

The demonstration system required two extensions to MICROEXPERT. Both extensions were made necessary by the 19 character limit on attribute and value length. The extensions are the procedures DEFINE and DISPLAY.

The procedure define allows the user to query the system for a definition or description of any valid rule response. The response definitions are stored in the default directory as ASCII text files with the extension

<.def>. The file name of each <.def> file is the first 8 characters of the response whose definition it contains. The 8 character limit is a restriction imposed by MS-DOS.

When called by the MICROEXPERT user interface, the procedure prompts the user for the response to be defined. It truncates the user input at 8 characters and then attempts to open the definition file. A successful file open causes the procedure to display the contents of the definition file on the CRT, 21 lines at a time. When the user desires the next 21 lines, he can press the carriage return. If the file open was unsuccessful, the message "Sorry, that definition is not available" is displayed and the procedure blocks waiting on a carriage return.

The procedure overcomes the 19 character limit on value names by allowing the knowledge engineer to explain their meaning in text files of unrestricted length, but requires the knowledge engineer to select value names that are unique in the first 8 characters. The definition not available provision frees the knowledge engineer from the responsibility of providing a definition file for names whose meanings are obvious as well as saving disk space.

The procedure is incorporated into the procedure "user_reply" which contains the function "numeric_test". The purpose of "numeric_test" is to classify user responses as legal or not legal. The classification is accomplished in a nested if structure within the function. The nested if is constructed so that each of the command strings (WHY, WHAT, HOW, RULE, QUIT, and now DEF) call the appropriate procedure which implements them. Inserting the procedure DEFINE consisted of expanding the nested if in "numeric_test" to include the command string DEF and installing the procedure DEFINE within the context of the procedure user_reply which contains the function numeric_test.

Testing for the procedure DEFINE was accomplished in isolation from MICROEXPERT by using a stub test driver to call DEFINE. The source code for DEFINE is provided in Appendix I.

The procedure DISPLAY is very similar to DEFINE except that it is designed to operate on the conclusion side of rules rather than their condition side. When an output of the system is too long to be contained in a 19 character value name, the rule which produces that output can be rewritten as follows:

```
if {conditions} then {attribute} is procedure  
display({filename})
```

Where "Filename" is the name of an ASCII text file in the default directory. Because of the MS-DOS restriction, filename is limited to no more than 8 characters. DISPLAY will automatically append the extension <.dat> to filename before attempting the open. DISPLAY behaves exactly DEFINE except that its "not found" message is "Output data file not present".

The procedure DISPLAY obviates the need for static lookup tables in the demonstration system and permits MICRDEXPERT to be utilized in situations requiring long output descriptions. The 8 character restriction imposed by MS-DOS is unfortunate but is needed for flexibility. The limitation was not a major factor in the construction of the demonstration system.

Testing of DISPLAY was accomplished in isolation using a stub driver similar to that used for DEFINE. Source code for DISPLAY is also provided in Appendix 1.

The design and recommended coding for DISPLAY can be found in chapter 9 of the MICRDEXPERT users manual. Installation instructions for DISPLAY are located there

as well. For reasons of style, I did not follow the coding recommendations precisely, but the differences in the two versions have no functional effect.

This chapter discussed the development, testing, and structure of the demonstration Expert System. The user instructions for the demonstration system are contained in the MICROEXPERT users manual. The rule base for meteorological consultations is contained in the disk file <meteo.kb> and provided in printed form in Appendix 2. A disk containing <meteo.kb> as well as the definition and output forecast files accompanies this paper as Exhibit 1.

The next chapter will discuss the final phase of the investigation, the extensibility analysis of MICROEXPERT. Investigation results are presented in Chapter 7 followed by recommendations for further work in Chapter 8.

CHAPTER 6

THE INVESTIGATION - EXTENSIBILITY ANALYSIS

Extensibility analysis presented a puzzling challenge. The objective was to determine how much effort was required to arbitrarily extend MICROEXPERT. One approach would have been to attempt some extensions and measure the time they required. However, such a course would give answers that were incomplete at best and misleading at worst. The validity of the estimates would depend entirely on the extensions selected for implementation. If MICROEXPERT had a structure that happened to easily accommodate the particular extensions selected, the effort estimates would be unreasonably low. On the other hand, if the extensions chosen for implementation were not representative of the type of modifications likely to be required in an engineering setting, any results would be meaningless.

The solution I chose was to attempt to port MICROEXPERT to an M68000-based machine. The purpose of this port was to determine how closely coupled the tool was to the IBM PC machine design and the Turbo Pascal implementation of the Pascal language standard. Because of its low cost, an Atari 1040 ST machine was selected as the target. Since Turbo Pascal is not available for

the Atari 1040 ST, Personal Pascal from Optimized Systems Software, San Jose, CA. was chosen as the development system. Personal Pascal implements many of the Turbo Pascal library calls while still remaining true to the Pascal language standard. In principle, if the port was an easy task, then MICROEXPERT was not closely coupled to the IBM PC and Turbo Pascal combination. Thus, it could be concluded that the program was easily portable to any standard Pascal environment and easily extensible once in that environment. If, on the other hand, the port was difficult or unsuccessful, it was reasonable to conclude the opposite; that MICROEXPERT was closely tied to a specific machine design and language implementation.

There were some non-technical consequences which resulted from selecting this method of investigation. A major obstacle to porting MICROEXPERT to the Atari 1040 ST is the media incompatibility between that machine and the IBM PC. MICROEXPERT is supplied on a 5.25 inch floppy disk formatted for the IBM PC. The assistance of a computer retailer was required to transfer the files to a 3.5 inch microfloppy compatible with the Atari 1040 ST. While media compatibility is not a significant technical problem, the media conversion was not easily

obtained and was performed on the condition that the retailer doing so not be identified.

Prior to compiling the original MICROEXPERT source code, I printed a listing and reviewed it. Some problems were immediately apparent. Turbo Pascal (at least through version 3.0) places a 64K limit on the size of any program module [11]. To counteract this restriction the Turbo Pascal compiler supports the use of program segmentation or memory overlaying [11]. In principle, the only portions of the program that must reside in local memory are that segment currently executing and any procedures which it references. The rest of the program may reside on the backing store.

Overlays are a variation on the same theme as virtual memory but with an important difference. In virtual memory systems, a reference to a logical address outside the current working set is translated by the Memory Management Unit (MMU) to the physical memory page on which the reference resides. That page is then swapped into the working set according to the MMU's page replacement algorithm. The MMU has knowledge of the entire program memory space in order to provide this service. In program overlay this is not the case. Each overlay is allocated its memory space as if it were a

stand-alone program. When the entire program is loaded, the main segment and any identifiers declared within its context are placed in low program memory (overlay 0). The first overlay segment (overlay 1) is loaded into memory immediately above overlay 0. Then execution of overlay 0 is begun. When a reference to overlay 1 is encountered the program context is shifted to overlay 1 along with program control.

Program context and control may shift back and forth between overlay 0 (sometimes called the base segment) and overlay 1 several times before any reference is made to overlay 2. When a reference to overlay 2 is made, overlay 2 is loaded into memory immediately above overlay 0 REPLACING OVERLAY 1. Overlay 2 has no access path to the context of Overlay 1. References to that context cause unpredictable errors.

The consequences of an overlay discipline for extensibility are significant. Extensions within the base segment present no real problem unless memory headroom is small. By expanding the size of the base segment, more total memory is required for the base and the overlay area. If the base and the overlay area are close to the current physical memory ceiling of the

machine, re-segmentation may be required. Extension in an overlay area causes an additional concern. References cannot be made across segment (overlay) boundaries except into the base segment (which always remains in memory). Thus, any identifiers from other segments that the extension requires, must be declared in the base segment and thereby be made visible to all contexts. Likewise, any identifiers that the extension wishes to make available to other segments must be declared in the base segment. These identifiers will also be visible to all contexts. Given the scoping rules of Pascal, identifier collision becomes a significant design issue.

Fortunately, the Atari 1040 ST has a 1 megabyte memory ceiling [4] and Personal Pascal does not support an overlay mechanism so the port was able to proceed when I removed the "overlay" keyword references from the original source. The next difficulty I noticed was within the user interface procedure that searches for the knowledge base file. That procedure uses MS-DOS calls which place the file names into Intel 8088 return registers and then prints the contents of those registers [8]. Thus the procedure is specific to MS-DOS and the IBM PC. It had to be replaced with an

equivalent procedure for the Atari 1040 ST and the M68000 architecture.

The Atari 1040 ST uses the iconographic user interface Graphics Environment Manager (GEM) from Digital Research Corporation [4]. This interface is similar to other icon based user interfaces that were popularized by the Apple MacIntosh. Calling MICROEXPERT from such an interface presents a problem since MICROEXPERT expects an MS-DOS command line. GEM provides no mechanism for entering a command line upon program invocation. Since lack of a command line argument causes the user to be prompted for the knowledge base file name, all that was necessary was to write a dummy interface procedure that provided MICROEXPERT with a null command line argument from GEM. This procedure replaced the MICROEXPERT procedure `get_command_line` in the original source.

At this point, I was able to clean up some annoying syntactical differences between the two Pascals and compile the code. The compile was unsuccessful. Analysis revealed that the problem was rooted in some dubious "enhancements" to the Pascal standard specific to Turbo Pascal. Turbo Pascal provides a library procedure `filichar` [11] which accepts an identifier

name, an integer expression, and one-byte (character) bit pattern. The procedure begins at the base address of the identifier (the identifier can have any type) and writes the byte pattern into memory for the next {integer expression} bytes. MICROEXPERT uses "fillchar" to initialize its rule representation structures and in its utility routines to perform string stuffing.

I cannot think of a more flagrant violation of Pascal's strong type structure [17] than fillchar. Personal Pascal did not support such a procedure [16]. I was able to code around the fillchar procedure using imbedded for and while loops but that solution is clumsy and slow.

Turbo Pascal provides another library procedure which violates Pascal's type system. The Turbo library procedure "val" [11] accepts an ASCII string and returns its numeric value as either a real number or an integer depending on which is appropriate. Expressions conforming to the real number grammar are returned as reals whereas those that conform to the more restrictive integer grammar are returned as integers. Turbo Pascal supports an inverse to "val" called "str" [11] which accepts either an integer or a real number as well as some format parameters. The function returns an ASCII

string representing the numeric input. MICROEXPERT uses these functions to represent numeric values in ASCII records. Since the rules are tagged with an ASCII representation of rule number, every rule base is affected not just those that utilize numeric attribute values.

I was able to code around this difficulty by separating "vai" and "str" into their real and integer components. Fortunately, the demonstration system did not involve real number processing so the integer versions of "vai" and "str" were adequate and could be substituted for the originals.

The final obstacle to the compile resulted from another curious provision of Turbo Pascal. A major criticism of Pascal's design has been the fact that static sizing was made part of the type declaration [17]. Thus, it is impossible to code a general string or array management procedure in standard Pascal. Turbo Pascal permits the programmer to invoke a compiler directive [11] which temporarily disables the sizing check called for by the Pascal standard. Disabling the check permits one to write a general string management procedure.

One codes such a procedure by defining a constant to stand for the string size in a type definition. That type can then be declared as the input string type for the utility string management procedure. In standard Pascal, the procedure could only accept strings of equal or lesser length than that specified by the type definition. The constant then becomes equivalent to the maximum value returned by the `length(string_variable)` function. Using Turbo Pascal's compiler directive, strings of any length are acceptable. The constant from the type definition retains its original value and is no longer related to string length.

MICROEXPERT makes use of this facility of Turbo Pascal in its string management utility routines. The compiler directive allows the utility procedures to be general and saves memory space. More importantly, the different interpretation of the length constant, allows for selective partitioning within long strings. For example, if a scanner is coded so that the input string is scanned from the first character to the `p`th character (`p` is a predefined constant in the type definition), the scanner will only scan the first `p` characters of any input string (regardless of its length). Under the rules of standard Pascal, the input string must have `p`

characters or less so the entire string will be scanned. In other words, partitioning is not permitted.

To compile the MICROEXPERT code on the Atari 1040 ST, I was forced to make all strings the same length. I accomplished this by changing the definition of the constant word_size to match the 80 character definition of an input line. That change has the effect of replacing the 19 character limit on attribute and value names with an 80 character limit. After making this and the other changes identified thusfar, the code compiled successfully.

The next step was to try to run the compiled program. When I attempted this, the program failed to parse the knowledge base file. Since parsing failed on several knowledge base files and analysis of these files showed that they conformed to the MICROEXPERT structure rules, my attention turned to the parser. I discovered that the MICROEXPERT parser explicitly scans for the IBM control character "etx" [8] as an end-of-file mark and classifies that character as a token. Scanning for an explicit end-of-file mark contradicts the Pascal notion of using the function EOF(<filename>) to check for an end-of-file condition.

Since the Atari 1040 ST does not use an end-of-file mark, [4] I replaced the etx definition with the null character (hexidecimal 0) and the "token = etx" conditions with EOF(input) conditions believing these substitutions to be equivalent. Despite these efforts, I was unable to get the parser to function and was compelled to abandon the port. I cannot determine whether the difficulty with the parser stems from changing the value of word_size, the lack of an etx token, or from some other difficulty that has yet to surface. In any case, the porting experience accomplished its goal which was to provide insight into the extensibility of MICROEXPERT.

When I abandoned the port, I had already invested significant programming effort in the task. I had been compelled to utilize facilities unique to Personal Pascal to emulate behavior unique to Turbo Pascal. I was facing a parser redesign at worst or a substantial re-coding of the parser at best. I was convinced that I had sufficient evidence to conclude that porting MICROEXPERT to the Atari 1040 ST was not a simple task. I could therefore conclude that MICROEXPERT was closely coupled to the Turbo Pascal implementation of Pascal and to the IBM PC machine design.

This chapter has detailed the attempted port of MICROEXPERT to an M68000 based machine architecture. This port was undertaken in an effort to determine the degree of machine specificity of the MICROEXPERT implementation. The porting effort was unsuccessful and it was concluded that this implementation of MICROEXPERT was indeed closely coupled to a particular language/machine combination. The next chapter discusses this and the other conclusions of the investigation in more detail.

CHAPTER 7 - CONCLUSIONS

This investigation consisted of building a demonstration Expert System that dealt with a problem representative of those that arise in a commercial engineering environment as well as an extensibility analysis of the development shell used to build that system. The investigation supports several conclusions about the current state of Expert System technology in the microcomputer environment.

The demonstration system demonstrates that it is possible to build a reasonably sophisticated Expert System for an acceptable manpower cost. While the S3 rules did not fully capture all the expertise available in the domain, they are more than adequate to handle all the data available to the target user population. Since heuristic problems in the commercial engineering sector have similar spanning problems within their domains, it is reasonable to infer that shells of this type could be used to produce worthwhile Expert Systems for engineering applications.

Unfortunately, MICROEXPERT is sufficiently machine specific to raise concern about the maturity of microcomputer resident Expert Systems. From the extensibility analysis, one can conclude that maximum

advantage was taken of the Turbo Pascal implementation of the Pascal standard. Considering the inherent inefficiency of the linked attribute list as a knowledge representation structure, these circumventions of the Pascal standard were probably required to maintain an acceptable level of program performance.

The extensibility analysis shows that MICROEXPERT is reasonably easy to extend with the IBM Turbo Pascal environment provided one remains wary of the overlay pitfall. Experience with overlays on other projects leads me to suspect that non-trivial extensions (such as incorporating fuzzy reasoning) will require re-segmentation of the program. Program re-segmentation is not an impossible task but it is definitely not a trivial one either.

The difficult questions arise when one considers the feasibility of introducing technology such as MICROEXPERT into the commercial engineering community. If one adopts the position that machine specificity is the price for performance, then Expert Systems enter the engineering center at an unacceptable cost. A typical commercial engineering environment contains a number of cooperating heterogeneous computer systems, each tailored for a specific set of engineering problems.

Since MICROEXPERT did not port easily into standard Pascal, it is unrealistic to presume that it could be used as a baseline shell for Expert System development on those types of tailored systems. Instead, the current practice of custom shells for each machine type will continue.

Nevertheless, the demonstration system has shown that it is possible to construct a useful Expert System given a general purpose shell. If such a shell could be implemented in a more portable language and freed from its architectural dependencies, the outlook for successful penetration of the commercial engineering environment is improved. The architectural dependencies seem to be the result of the inherent inefficiencies of the linked attribute list as a representation structure. The shell is forced to construct, maintain, and traverse a number of such lists during execution. To accomplish these tasks efficiently, machine dependent optimizations are necessary.

Further, intrinsic type conversion facilities are needed in Expert Systems. These functions overcome the limitations imposed by requiring object storage to be statically allocated at compile time. This requirement forces the programmer to select a single type (or a

fixed set of types) for the elements in the knowledge representation structure. If the particular inference technique used operates on objects of a different type than any of those specified, the conversion must be performed dynamically.

MICROEXPERT makes good use of the facilities of Turbo Pascal to overcome these limitations. In doing so, however, the program sacrifices portability since the Turbo Pascal facilities circumvent the Pascal standard at a fundamental level. An implementation in a language such as "C" which does not have Pascal's restrictive structure may produce a product with acceptable performance and portability. Such a product stands a reasonable chance of gaining wide acceptance in the commercial engineering community.

CHAPTER 8 - RECOMMENDATIONS FOR FURTHER WORK

While this investigation attained its goals, there are a number of interesting questions that it did not answer. In addition, the data that it did provide raised a number of significant questions as well. These problems are posed in this chapter. The intent of this chapter is to suggest areas for future investigation.

The demonstration system showed that it is possible to construct an adequate knowledge base for limited weather prediction. However, the knowledge base captures only a small fraction of the heuristics available from meteorology. Further, the discipline of meteorology has been unable to develop an efficient knowledge representation schema of its own. Much work remains to be done in the area of meteorological heuristics development and knowledge representation, particularly in regard to the problem of structuring large quantities of data for rapid processing.

The Watts inference technique is most effective when it employs visual templates of sky observations. In principle, an icon-based user interface (such as the one on the Apple MacIntosh) could accomodate such templates. The limiting design constraints on such a system would be efficient template storage in memory,

the template entry mechanism (how does the developer place a template into the system?), and the template-to-rule base interface (what does the rule base do when a user selects a template?).

Finally, there is the implementation of a standard shell in "C" referred to at the end of the previous chapter. Any such system, even if it were merely a transcription of the MICROEXPERT code could be an interesting research tool for analyzing the inherent machine dependencies common to Expert Systems.

Bibliography

1. Aherns, C. Donald, "Meteorology Today", copyright 1982, West Publishing Company, New York, NY
2. Brachman, Ronald J., "The Basics of Knowledge Representation and Reasoning", AI&I Technical Journal, January-February 1988, Volume 67, Issue 1
3. Campbell, Tim, "The Progressive Farmer (R) Do-It-Yourself Weather Book", copyright 1979, Oxmoor House Inc., Birmingham, AL
4. Gertls, K., Engiisch L., and Bruckman R., "Atari ST Internals", copyright 1988, Abacus Software Inc., Grand Rapids, MI
5. Hirschberg, Julia, Bailard, Bruce W., and Hindle, Donald, "Natural Language Processing", AI&I Technical Journal, January-February 1988, Volume 67, Issue 1
6. Hunt, V. Daniel, "Artificial Intelligence and Expert Systems Sourcebook", copyright 1986, Chapman and Hall Inc.
7. Mishkoff, Henry C., "Understanding Artificial Intelligence", copyright 1969, Howard W. Sams and Company
8. Norton, Peter, "Inside the IBM PC", copyright 1986, Brady Books Inc.
9. Pettersen, Sverre, "Introduction to Meteorology", copyright 1969, McGraw-Hill Inc.
10. Rauch-Hindin, Wendy B., "Artificial Intelligence in Business, Science, and Industry", copyright 1986, Prentice-Hall Inc.

11. Swan, Tom, "Mastering Turbo Pascal", copyright 1986, Hayden Book Company
12. Thompson, Beverly and Thompson William, "Microexpert Version 1.0 - IBM PC", copyright 1985, McGraw-Hill Inc.
13. Tong, Richard M., Neuman, Neville D., Berg-Cross, Gary, and Rook, Fred, "Performance Evaluation in Artificial Intelligence Systems", DARPA Field Technical Report, August 17, 1987, Contract No. DAAH01-86-C-1053, ARPA Order No. 5916
14. Watts, Alan, "Instant Weather Forecasting", copyright 1968, Adlard Coles Ltd., London, UK
15. Watts, Alan, "Weather Forecasting Ashore and Afloat", copyright 1967, Adlard Coles Ltd., London, UK
16. Wilkinson, Bill and Rice, Earl, "A Reference Manual for Personal Pascal - Atari 520/1040 ST Version 2", copyright 1987, Optimized Systems Software Inc., San Jose, CA
17. Writh, Nicklaus and Jensen, Kathleen, "Pascal User's Manual and Report", 2nd Edition, copyright 1974, Springer-Verlag Inc.

Appendix I
Source Code for Extensions

Display

```
PROCEDURE Display(parameter: parm_array);
VAR
  f: text;
  text_string: string80;
  ch: char;
  error, count: integer;
BEGIN (* begin procedure display *)
  count:=0;
  (*$I-*)
    assign(f, parameter[1]);
    reset(f);
    error:=ioresult;
  (*$I+*)
  if (error=0) then
    BEGIN
      readln(f, text_string);
      WHILE not EOF(f) DO
        writein(text_string);
        count:=count+1;
        if (count=21) then
          BEGIN
            read(ch);
            count:=0;
          END;
        readln(f, text_string);
      END; (* while loop *)
      close (f)
    END (* if error=0 *)
  else
    writeln('The requested forecast file is not
    available');
    writeln;
    writeln;
    writein('Press <Enter> or <Return> to
    continue');
    read (ch)
  END; (* procedure display *)
```

Define

```
PROCEDURE Define;
(* Prompts the user for the term to define in      *)
(* response to the "DEF" command. Truncates the    *)
(* term given to 8 chars (for MS-DOS) and appends  *)
(* the extension ".def". Searches for a text file  *)
(* with that name. If found, displays its contents *)
(* 21 lines at a time. If NOT found, prints a      *)
(* "definition not available" message.             *)
(*                                                  *)
(*                                                  *)
(*                      DCW - 06/27/88              *)
(*                                                  *)
VAR
file_name,nput:word;
t_name:string[8];
text_string:string80;
ch:char;
f:text;
count,error:integer;

BEGIN
count:=0;
Writeln('Term to Define >');
Readln(nput);
clrscr;
writeln;
t_name:=copy(nput,1,8);
file_name:=concat(t_name,'.def');
(*$I-*)
    assign(f,file_name);
    reset(f);
    error:=ioresult;
(*$I+*)
if (error=0) then
    BEGIN
    readln(f,text_string);
    WHILE NOT eof(f) DO
        BEGIN
        Writein(text_string);
        count:=count+1;
        if (count=21) then
            BEGIN
            read(ch);
            count:=0;
            END; (* if then *)
        readln(f,text_string);
        END; (* while loop *)
    close(f)
```

```

      END (* if then begin *)
    else
      WriteIn(' A definition for that term is not
available');
      WriteIn;
      WriteIn;
      WriteIn('Press <Enter> or <Return> to continue');
      read(ch)
    END; (* procedure define *)

```

Notes: These procedures assume the existence of the predefined types word, text, and string80 as described in the MICROEXPERT documentation. Further, they are written in Turbo Pascal (a product of Bordland Inc.) and require Turbo Pascal library procedures.

Appendix II
The Rule Base Meteo.kb

Note 1) For esthetic reasons, the rule base file has been reformatted for publication in this appendix. See the MICROEXPERT documentation for instructions on the required format for knowledge base files

Note 2) Am_meteo is a null goal. Its function in the rule base is to permit the formation of guard rules that allow the inference engine to handle section transitions.

RULES

- 1) if clouds_visible is yes and know_type is yes then
am_meteo is yes.
- 2) if clouds_visible is yes and know_type is no and
cloud_group is low and cloud_shape is heaped then
cloud_type is cumulus.
- 3) if clouds_visible is yes and know_type is no and
cloud_group is low and cloud_shape is
heaped_&_layered then cloud_type is stratocumulus.
- 4) if clouds_visible is yes and know_type is no and
cloud_group is vertical_extenders and cloud_shape is
heaped then cloud_type is cumulonimbus.
- 5) if clouds_visible is yes and know_type is no and
cloud_group is low and cloud_shape is amorphous then
cloud_type is stratus.
- 6) if clouds_visible is yes and know_type is no and
cloud_group is low and cloud_shape is layered then
cloud_type is nimbostratus.
- 7) if clouds_visible is yes and know_type is no and

cloud_group is vertical_extenders and cloud_shape is layered then cloud_type is nimbostratus.

- 8) if clouds_visible is yes and know_type is no and cloud_group is high and cloud_shape is wispy then cloud_type is cirrus.
- 9) if clouds_visible is yes and know_type is no and cloud_group is high and cloud_shape is heaped then cloud_type is cirrocumulus.
- 10) if clouds_visible is yes and know_type is no and cloud_group is high and cloud_shape is amorphous then cloud_type is cirrostratus.
- 11) if clouds_visible is yes and know_type is no and cloud_group is medium and cloud_shape is heaped then cloud_type is altocumulus.
- 12) if clouds_visible is yes and know_type is no and cloud_group is medium and cloud_shape is layered then cloud_type is altostratus.
- 13) if clouds_visible is no and surface_wind is no and cross_winds is unknown then forecast is procedure display('w14.dat').
- 14) if clouds_visible is no and surface_wind is yes then forecast is procedure display('w24.dat').
- 15) if cloud_type is cirrostratus and visibility is poor and cross_winds is worsen and cloud_mass is organized then forecast is procedure display('w2s.dat').
- 16) if cloud_type is cirrostratus and visibility is poor and cross_winds is worsen and cloud_mass is disorganized then forecast is procedure display('w2m.dat').

- 17) if cloud_type is altocumulus and humidity is high and cross_winds is worsen and cloud_mass is splotchy then forecast is procedure display('w7s.dat').
- 18) if cloud_type is cirrus and cross_winds is worsen and visibility is good then forecast is procedure display('w1s.dat').
- 19) if cloud_type is cirrus and cross_winds is worsen and visibility is poor then forecast is procedure display('w1m.dat').
- 20) if cloud_type is cirrostratus and visibility is poor and cloud_mass is disorganized then forecast is procedure display('w2mid.dat').
- 21) if cloud_type is altocumulus and cross_winds is worsen and cloud_mass is splotchy then forecast is procedure display('w7m.dat').
- 22) if cloud_type is stratocumulus and cloud_mass is widespread and humidity is high then forecast is procedure display('w9m.dat').
- 23) if cloud_type is altostratus and cross_winds is worsen then forecast is procedure display('w3m.dat').
- 24) if cloud_type is cumulus and ratio is large then forecast is procedure display('w1is.dat').
- 25) if cloud_type is cumulus and ratio is small and cloud_mass is line then forecast is procedure display('w12m.dat').
- 26) if cloud_type is cumulus and ratio is large and cloud_mass is separated then forecast is procedure display('w13m.dat').

- 27) if cloud_bank is east and cloud_mass is east_only
and front_past is cold then forecast is procedure
display('w15.dat').
- 28) if cloud_bank is west and cloud_mass is west_only
and front_past is warm then forecast is procedure
display('w15A.dat').
- 29) if cloud_type is altocumulus and cloud_mass is
banded_broken and cross_winds is unknown then
forecast is procedure display('w17.dat').
- 30) if cloud_type is cumulus and ratio is small and
cloud_mass is insular then forecast is procedure
display('w19.dat').
- 31) if cloud_type is cirrus and cross_winds is improve
and pressure_fail is no then forecast is procedure
display('w20.dat').
- 32) if cloud_type is stratocumulus and cloud_mass is
patchy and pressure_fail is no then forecast is
procedure display('w21.dat').
- 33) if cloud_type is cirrus and cross_winds is unknown
then forecast is procedure display('w1mid.dat').
- 34) if cloud_type is cirrus and cross_winds is worsen
then forecast is procedure display('w3m.dat').
- 35) if cloud_type is altostratus and cloud_mass is
thickening then forecast is procedure
display('w4m.dat').
- 36) if cloud_type is altostratus and cloud_mass is thick
then forecast is procedure display('w5m.dat').
- 37) if cloud_type is stratus and cloud_mass is breaking
then forecast is procedure display('w6m.dat').

- 38) if cloud_type is stratus and cloud_mass is lighter
then forecast is procedure display('w6mld.dat').
- 39) if cloud_type is cumulus and cloud_mass is roiling
then forecast is procedure display('w8m.dat').
- 40) if cloud_type is cirrus and cross_winds is improve
then forecast is procedure display('w16.dat').
- 41) if cloud_type is stratus and cloud_mass is lumpy
then forecast is procedure display('w18m.dat').
- 42) if cloud_type is stratus and humldlty is high then
forecast is procedure display('w22.dat').
- 43) if cloud_type is altocumulus and cloud_mass is
speckied then forecast is procedure
display('w23.dat').
- 44) if cloud_type is stratus and humidity is low then
forecast is procedure display('w24.dat').
- 45) if surface_wind is yes then am_meteo is yes.
- 46) if surface_wind is no then cross_winds is unknown.
- 47) if upr_wnd_trk is left_to_right then cross_winds is
worsen.
- 48) if upr_wnd_trk is rlight_to_left then cross_winds is
lmprove.
- 49) if upr_wind_trk is not_known then cross_winds is
unknown.
- 50) if rcnt_temp_trnd is rlse then front_past is warm.

- 51) if rcnt_temp_trnd is fall then front_past is cold.
- 52) if upr_wnd_trk is back_to_front then cross_winds is no_change.
- 53) if upr_wnd_trk is front_to_back then cross_winds is no_change.

PROMPTS

prompt {clouds_visibile} Are there any clouds visibile?

prompt {know_type} Do you know the type?

prompt {cloud_group} Is the cloud group low, medium, high, or vertical_extenders?

prompt {cloud_shape} What is the shape (stricture) of the clouds?

prompt {visibiilty} Exclusive of man-made effects, is the visibiilty poor or good?

prompt {cloud_mass} How would you describe the overall state of the clouds? --Hint-- use the def command to see an explanation of some of these terms.

prompt {humidity} What is the humidity?

prompt {ratio} What is the ratio of the height of the cumuius clouds to their width?

prompt {surface_wind} Is there a surface wind biowing?

prompt {cloud_bank} In which direction is the cloud bank?

prompt {pressure_fall} Did (or is) the barometer falling?

prompt {upr_wnd_trk} With your back to the lower wind, in which direction is the upper wind tracking? (watch the movement of the streaky high clouds).

prompt {rcnt_temp_trnd} What did the temperature do the last time it changed?

Translations

trans {clouds_visible} Clouds can /not/ be seen

trans {know_type} You did /not/ know the type

trans {cloud_group} The Howard System cloud group

trans {cloud_shape} The structure of the clouds

trans {cloud_type} The type of clouds

trans {visibility} The visibility is /not/ good

trans {cross_winds} The weather trend indicated by the "crossed winds" rule is

trans {cloud_mass} The general character of the cloud mass is

trans {humidity} The humidity level is

trans {ratio} The ratio of the height of the cumulus clouds to their width is

trans {surface_wind} There was /not/ a surface wind

trans {front_past} The front that just passed is

trans {pressure_fall} The barometer did /not/ fall

trans {upr_wnd_trk} The direction of the upper wind relative to the lower wind is

trans {rcnt_temp_trnd} The most recent temperature trend was

AN EXPERT SYSTEM FOR SHORT-TERM WEATHER FORECASTING

by

David Charles Willett

B.S., Georgia Institute of Technology, 1977

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Expert Systems are computer programs which emulate the behavior of human experts by using an inference mechanism to operate on a stored knowledge base. These systems show great promise of making the expertise of scarce human specialists more generally available. It has been stated that while expert systems are not yet ready to replace human experts, several systems have been of value in assisting non-expert specialists to perform at expert level.

This project investigates the nature of expert systems by developing a small one which can be run on a microcomputer. The system was developed using a commercial development tool known as a "shell" and is targeted to run on an AT&T PC 6300, Zenith Data Systems 150, or any Intel 8088 family machine. The system will support a limited vocabulary of English-like responses. Further, the system will support extension in the knowledge base and user interface.

Expert Systems are appropriate for problems meeting specific criteria of non-determinism, complexity, and tractability. These problems are not trivial, but are amenable to heuristic attack. Further, there is a recognized body of human expertise concerning them available.

The problem of short term forecasting of local weather patterns meets the above criteria. The problem is sufficiently complex to require the application of specialized skills, and there exist accepted inference methods for its solution. In addition, to predict the near future weather successfully requires the use of evidence from several sources. The individual pieces of evidence, however, can easily be gathered by the average person.